НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»
Завідувач кафедри
_____ Наталія АУШЕВА
“___”_____2026 р.

Дипломна робота
на здобуття ступеня бакалавр

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “ Інформаційно-комунікаційна система обліку клієнтів підприємства”

Виконав: студент 4 курсу, групи ТР-21
Байдала Артем Миколайович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник ст.викладач Ольга БЕСПАЛА ____ _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль ст.викладач Ольга БЕСПАЛА _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ - 2026

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА
(підпис)
“___” _____2026 р.

ЗАВДАННЯ
на дипломну роботу студенту

Байдали Артема Миколайовича
(прізвище, ім’я, по батькові)

1. Тема роботи “ Інформаційно-комунікаційна система обліку клієнтів підприємства (веб-застосунок) ”
Науковий керівник Беспала Ольга Миколаївна, ст.викладач
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “28” травня 2026 року № 1992-с

2. Термін подання студентом роботи 05.06.2026

3. Вихідні дані до роботи мова програмування Python, фреймворк Django, СУБД, середовище розробки Visual Studio Code

4. Перелік питань, які потрібно розробити _____

1) провести аналіз серед наявних аналогів;

2) визначити головні функції веб-застосунку, що керує послугами та ресурсами;

3) аргументувати вибрані інструменти, алгоритми та технології для реалізації веб-системи;

4) побудувати архітектуру програмного забезпечення та бази даних; 5) створити прикладний програмний веб-інтерфейс;

6) представити процес застосування веб-інтерфейсу.

5. Орієнтований перелік ілюстративного матеріалу схема, що показує роботу фреймворку Django, схема архітектури проєкту, ER-діаграма класів бази даних, діаграма прецедентів (ролей), скріншоти роботи користувачів з веб-застосунком.

6. Дата видачі завдання 05.01.2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	10.12.2025 – 05.01.2026	Виконано
2.	Аналіз методів та засобів розв’язання задачі	06.01.2026 – 01.02.2026	Виконано
3.	Розробка архітектури та загальної структури системи	02.02.2026 – 09.03.2026	Виконано
4.	Розробка окремих підсистем	09.03.2026 – 30.03.2026	Виконано
5.	Програмна реалізація системи	30.03.2026 – 08.05.2026	Виконано
6.	Оформлення пояснювальної записки	11.05.2026 – 15.05.2026	Виконано
7.	Захист програмного забезпечення	13.05.2026	Оцінка 80/100
8.	Передзахист	03.06.2026 р.	Допущено
9.	Захист	Червень 2026 р.	

Студент _____ Артем БАЙДАЛА
(підпис) (прізвище та ініціали)

Керівник _____ Ольга БЕСПАЛА
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана на 74 сторінках, містить 15 ілюстрацій, 13 таблиць, 2 додатки, 37 джерел в переліку посилань.

Мета роботи — проєктування та реалізація інформаційно-комунікаційної системи у вигляді веб-застосунку для автоматизації обліку клієнтів та управління процесом надання консультаційних послуг з кібербезпеки в умовах сервісної IT-компанії.

Методи та засоби: клієнт-серверна архітектура з використанням архітектурного патерну MVC (Model-View-Controller); рольова модель доступу (RBAC) для забезпечення конфіденційності даних; реляційне моделювання баз даних; мова програмування Python та веб-фреймворк Django; система управління базами даних PostgreSQL (або MySQL); інструменти фронтенд-розробки HTML5, CSS3 та JavaScript; об'єктно-орієнтоване проєктування з використанням мови UML (діаграми прецедентів, класів та послідовностей).

Результат — інформаційно-комунікаційна система «my_crm» (веб-застосунок), яка забезпечує централізоване зберігання даних, розмежування прав доступу користувачів (Адміністратор, Менеджер, Клієнт) , автоматизований облік та контроль статусів обробки заявок на IT-послуги та консультації з кібербезпеки.

Ключові слова: ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНА СИСТЕМА, ВЕБ-ЗАСТОСУНОК, CRM-СИСТЕМА, ОБЛІК КЛІЄНТІВ, КІБЕРБЕЗПЕКА, РОЛЬОВА МОДЕЛЬ ДОСТУПУ, DJANGO, PYTHON.

ABSTRACT

The Bachelor's thesis consists of 74 pages, contains 15 illustrations, 13 tables, 2 appendix, and 37 references in the bibliography.

The objective of the thesis is the design and implementation of an information and communication system in the form of a web application to automate client accounting and manage the workflow of providing cybersecurity consulting services within a service-based IT company.

Methods and tools used: client-server architecture based on the MVC (Model-View-Controller) architectural pattern; Role-Based Access Control (RBAC) model to ensure data confidentiality ; relational database modeling; Python programming language and Django web framework; PostgreSQL (or MySQL) relational database management system; Front-end development tools including HTML5, CSS3, and JavaScript; object-oriented design using UML (Use Case, Class, and Sequence diagrams).

The result is the "my_crm" information and communication system (web application) that provides centralized data storage, user role differentiation (Administrator, Manager, Client) , automated accounting, and real-time status tracking for IT service requests and cybersecurity consultations.

Keywords: INFORMATION AND COMMUNICATION SYSTEM, WEB APPLICATION, CRM SYSTEM, CLIENT ACCOUNTING, CYBERSECURITY, ROLE-BASED ACCESS CONTROL, DJANGO, PYTHON.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ІКС – інформаційно-комунікаційна система.

ПЗ – програмне забезпечення.

БД – база даних.

СУБД – система управління базами даних.

ORM (Object-Relational Mapping) – технологія об'єктно-реляційного відображення, що забезпечує взаємодію між об'єктами програми та таблицями бази даних.

MVC (Model-View-Controller) – архітектурний шаблон побудови програмних систем, який передбачає розділення моделі даних, інтерфейсу та логіки керування.

MVT (Model-View-Template) – архітектурний шаблон, що використовується у фреймворку Django.

RBAC (Role-Based Access Control) – модель керування доступом на основі ролей користувачів.

UML (Unified Modeling Language) – уніфікована мова моделювання програмних систем.

ER-діаграма (Entity-Relationship Diagram) – діаграма сутностей і зв'язків бази даних.

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертекстових даних.

HTTPS (HyperText Transfer Protocol Secure) – захищена версія протоколу HTTP.

API (Application Programming Interface) – програмний інтерфейс взаємодії між програмними компонентами.

SQL (Structured Query Language) – мова структурованих запитів до баз даних.

SQL Injection (SQL-ін'єкція) – тип атаки, спрямований на виконання несанкціонованих SQL-запитів.

XSS (Cross-Site Scripting) – атака, що полягає у впровадженні шкідливого сценарію у веб-сторінку.

CSRF (Cross-Site Request Forgery) – міжсайтова підробка запитів.

UI (User Interface) – користувацький інтерфейс.

CRUD (Create, Read, Update, Delete) – базові операції створення, перегляду, редагування та видалення даних.

ВСТУП

Актуальність теми. Сучасний етап розвитку інформаційних технологій характеризується постійним розширенням цифрового середовища та зростанням ролі інформаційних ресурсів у діяльності підприємств. Для забезпечення стабільної роботи організацій виникає потреба в інструментах, які дозволяють ефективно накопичувати, опрацьовувати та передавати дані між користувачами без обмежень за місцем доступу чи часом використання.

У діяльності сервісних компаній питання організації роботи з клієнтами набуває особливого значення, оскільки якість обслуговування напряму залежить від швидкості обробки звернень та доступності інформації. Використання неструктурованих способів ведення обліку, окремих локальних файлів або ручного опрацювання даних ускладнює контроль за виконанням завдань, збільшує ймовірність втрати інформації та знижує загальну ефективність внутрішніх процесів.

Додаткові вимоги виникають у сфері IT-консалтингу та консультацій із кібербезпеки, де обробляються відомості з підвищеним рівнем чутливості. У таких умовах важливо не лише забезпечити зручність роботи із заявками клієнтів, а й реалізувати механізми контролю доступу, розмежування повноважень і захисту інформаційних ресурсів від несанкціонованого використання.

Існуючі універсальні системи управління взаємодією з клієнтами не завжди можуть бути ефективно застосовані для вузькоспеціалізованих завдань через складність адаптації, надлишковий функціонал або значні витрати на впровадження. У зв’язку з цим актуальним є створення спеціалізованої інформаційно-комунікаційної системи, орієнтованої на автоматизацію обліку клієнтів, централізоване керування заявками та підтримку процесів надання консультаційних послуг у сфері кібербезпеки.

Метою дипломної роботи є проєктування та реалізація інформаційно-комунікаційної системи у вигляді веб-застосунку для автоматизації обліку

клієнтів та управління процесом надання консультаційних послуг з кібербезпеки в умовах сервісної IT-компанії.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати предметну область, дослідити процеси взаємодії з клієнтами в сервісних IT-організаціях та обґрунтувати актуальність розробки системи.

2. Виконати порівняльний аналіз існуючих аналогічних рішень (універсальних таблиць, хмарних сервісів, комерційних CRM-систем) та виявити їхні недоліки для обраної ніші.

3. Розглянути та обґрунтувати вибір методів, архітектурних підходів (клієнт-серверна архітектура, патерн MVC) та моделей організації доступу до даних.

4. Обґрунтувати вибір технологічного стеку (мов програмування, веб-фреймворків, реляційної системи управління базами даних та Front-end інструментів) для реалізації веб-застосунку.

5. Спроєктувати загальну архітектуру системи та структуру реляційної бази даних із побудовою відповідних UML-діаграм (діаграм прецедентів, класів та послідовностей).

6. Програмно реалізувати веб-застосунок із модулями обробки даних клієнтів, обліку заявок на IT-послуги та гнучкою системою розмежування прав доступу на основі ролей.

7. Провести тестування створеного програмного продукту на відповідність функціональним вимогам, перевірити стабільність його роботи та безпеку доступу.

Об'єктом дослідження є процеси реєстрації, зберігання, редагування, перегляду інформації про клієнтів та їхні запити на отримання послуг у сервісних IT-компаніях.

Предметом дослідження є методи, моделі, архітектурні підходи та програмні засоби створення веб-орієнтованих інформаційно-комунікаційних систем для забезпечення ефективної взаємодії між менеджерами та клієнтами.

Обмеження системи. Розробляється система внутрішнього обліку, яка не охоплює фінансові розрахунки, транзакції чи бухгалтерську звітність. Роль кібербезпеки в межах цього проєкту є виключно консультативною (облік заявок на аудит, консультації тощо) і не передбачає написання спеціалізованого програмного коду для захисту мережевої інфраструктури. Доступ до функціоналу системи здійснюється виключно через веб-браузер.

Апробація результатів роботи. Основні положення та результати розробки інформаційно-комунікаційної системи були представлені, обговорені та отримали схвалення на XI Міжнародній науково-практичній конференції студентів, аспірантів та молодих вчених «Цифрові технології в енергетиці та кібербезпека» (м. Київ, НТУУ «КПІ ім. Ігоря Сікорського», 14–15 травня 2026 р.). За матеріалами дослідження опубліковано тези доповіді у збірнику наукових праць.

Структура та обсяг роботи. Дипломна робота складається зі вступу, чотирьох розділів основної частини, висновків, списку використаних джерел та додатків.

У першому розділі наведено постановку прикладної задачі, детальний огляд предметної області, а також аналіз існуючих програмних аналогів. Другий розділі присвячено аналізу інженерних методів, архітектурних підходів та обґрунтуванню моделей організації баз даних і доступу. У третьому розділі виконано проєктування системи, описано структуру бази даних та наведено UML-діаграми. Четвертий розділ містить опис програмної реалізації, інструкцію користувача та результати тестування системи.

Загальний обсяг текстової частини роботи становить 74 сторінки. Робота містить 15 рисунків , 13 таблиць та 2 додатків. Список використаних джерел включає 37 найменувань.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТОНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

У цьому розділі розглядається сутність процесів взаємодії з клієнтами в умовах сучасної сервісної IT-компанії. Визначено ключові етапи обробки запитів на консультаційні послуги з кібербезпеки, проаналізовано інформаційні потоки та виявлено слабкі місця в існуючих неавтоматизованих підходах до ведення обліку. На основі проведеного аналізу сформульовано передумови для створення спеціалізованої інформаційно-комунікаційної системи.

1.1 Опис предметної області та аналіз процесів взаємодії з клієнтами

Предметною областю даного дослідження є процеси операційного обліку, реєстрації, обробки та моніторингу заявок користувачів у компаніях, що спеціалізуються на наданні професійних IT-послуг [1]. Як об'єкт автоматизації розглядається сервісна IT-компанія, діяльність якої зосереджена на наданні консультаційних послуг у сфері інформаційної безпеки, проведенні аудитів захищеності для малого бізнесу та наданні технічної підтримки веб-ресурсів.

Взаємодія між сервісною компанією та її клієнтами є багатокроковим процесом, ефективність якого безпосередньо впливає на лояльність замовників та рентабельність бізнесу. Узагальнений життєвий цикл клієнтської заявки в межах предметної області складається з таких ключових етапів (див. рис. 1.1):

1. Ініціація звернення. Потенційний або існуючий клієнт формує запит на отримання консультації (наприклад, проведення первинного аналізу вразливостей сайту, налаштування політик безпеки або розслідування інциденту).

2. Первинна реєстрація та класифікація. Менеджер компанії фіксує контактні дані замовника, визначає категорію складності та специфіку запиту.

3. Обробка та виконання. Запит передається профільному фахівцю. На цьому етапі відбувається безпосереднє надання консультаційної послуги, формування рекомендацій та визначення термінів усунення ризиків.

4. Закриття заявки та архівування. Менеджер фіксує статус «Виконано», а історія взаємодії зберігається в інформаційному контурі підприємства для подальшого аналізу.

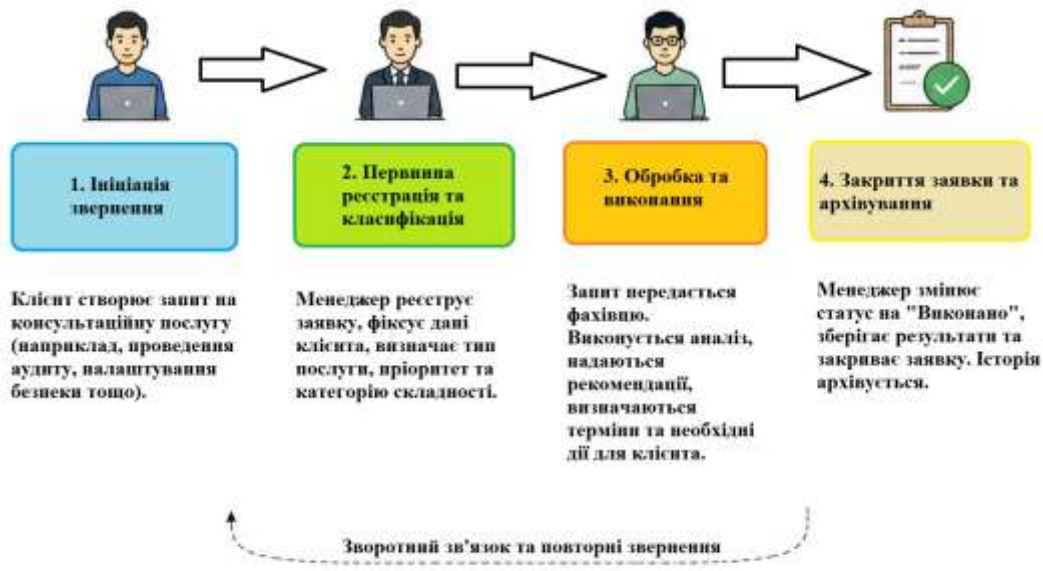

Рисунок 1.1 – Схема процесу обробки заявки клієнта

Аналіз поточної практики організації цих процесів у компаніях малого та середнього бізнесу свідчить про використання розрізнених і часто застарілих інструментів. Документообіг та облік ведеться за допомогою локальних електронних таблиць (наприклад, Microsoft Excel) або паперових носіїв. Такий підхід породжує низку критичних проблем та неефективностей:

- Втрата інформації та «людський фактор». За відсутності єдиної централізованої бази даних профілі клієнтів або статуси їхніх звернень можуть дублюватися, втрачатися або некоректно оновлюватися різними співробітниками підприємства.
- Низька швидкість реакції. Менеджери змушені вручну перевіряти наявність нових запитів із різних каналів зв'язку, що суттєво сповільнює процес обробки критично важливих заявок та збільшує час очікування відповіді.

- Відсутність прозорості для клієнта. Замовник не має можливості в режимі реального часу відстежувати, на якому етапі виконання перебуває його запит, що змушує його здійснювати додаткові комунікації (дзвінки, листи).
- Ризики конфіденційності. Консультаційні послуги з кібербезпеки передбачають роботу із чутливою інформацією про архітектуру мереж клієнта чи виявлені дірки в захисті. Використання незахищених локальних файлів без розмежування прав доступу створює пряму загрозу витоку цих даних.

З погляду інформаційних потоків, предметна область характеризується постійним обміном даними між трьома основними суб'єктами: адміністратором системи, операційним менеджером та безпосередньо клієнтом. Кожен із цих суб'єктів генерує та споживає специфічні масиви інформації.

Впровадження автоматизованої інформаційно-комунікаційної системи (ІКС) покликане об'єднати ці потоки в єдиний безпечний веб-інтерфейс [2]. Система має забезпечити чітку спеціалізацію: замість ускладненого комерційного функціоналу загальних CRM-систем (маркетинг, лідогенерація, фінансові розрахунки), фокус зміщується на оперативний облік клієнтів, безпечне зберігання профілів та прозорий контроль за зміною статусів заявок на IT-консультації. Це дозволить мінімізувати час на адміністрування процесів, структурувати внутрішню базу знань компанії та гарантувати клієнтам високий рівень конфіденційності.

1.2 Обґрунтування актуальності автоматизації обліку та консультаційних послуг

Ефективність функціонування будь-якої сервісної організації в сучасних умовах безпосередньо залежить від рівня автоматизації її внутрішніх бізнес-процесів. Перехід від традиційних методів ведення обліку (локальні файли, паперові носії, розрізнені текстові документи) до спеціалізованих інформаційно-комунікаційних систем є не просто технологічним трендом, а об’єктивною інженерною та економічною необхідністю [3]. Це зумовлено декількома

критичними факторами, які визначають життєздатність компанії на ринку IT-послуг.

По-перше, головною технічною передумовою є усунення проблеми розрізненості даних (так званих «інформаційних ізолятів»). Коли інформація про клієнтів та їхні поточні запити зберігається на локальних комп'ютерах різних менеджерів, виникає висока ймовірність дублювання, неузгодженості та втрати актуальних відомостей. Автоматизація дозволяє створити єдину реляційну базу даних із централізованим керуванням [4]. Це гарантує цілісність, несуперечність та постійну доступність інформації для всіх авторизованих співробітників у режимі реального часу, незалежно від їхнього фізичного розташування, що є критично важливим в умовах поширення дистанційної та гібридної форм роботи.

По-друге, автоматизація процесів обліку кардинально знижує вплив «людського фактора» на операційну діяльність. Застосування інформаційної системи дозволяє автоматизувати рутинні операції, такі як:

- реєстрація нових користувачів та автоматичне присвоєння унікальних ідентифікаторів;
- фіксація точного часу створення заявки та автоматичний контроль термінів її обробки;
- валідація введених даних на етапі заповнення форм, що запобігає появі некоректних або неповних записів.

Особливого значення автоматизація набуває у контексті надання консультаційних послуг з кібербезпеки. Оскільки цей напрямок діяльності безпосередньо пов'язаний із конфіденційною інформацією про стан захищеності інформаційних систем замовників (наявність вразливостей, архітектурні недоліки мереж, інциденти безпеки), збереження цих даних вимагає підвищеного рівня надійності. Звичайні офісні пакети чи загальнодоступні хмарні таблиці не здатні забезпечити гнучкого та надійного розмежування прав доступу.

Впровадження спеціалізованої ІКС дозволяє реалізувати рольову модель доступу (Role-Based Access Control, RBAC) [5]. Завдяки цьому адміністратор системи може чітко розмежувати повноваження (див. в рис. 1.2): клієнт бачить

виключно власні заявки та історію своїх консультацій; менеджер отримує доступ до обробки та призначення заявок у межах своєї компетенції; системні конфігурації залишаються доступними лише для адміністратора. Такий підхід мінімізує ризики внутрішніх витоків інформації та забезпечує відповідність базовим вимогам конфіденційності.

Крім того, автоматизація комунікаційної складової (обробка статусів заявок у веб-застосунку) дозволяє структурувати взаємодію між клієнтом та виконавцем. Замість хаотичних дзвінків та листування в різних месенджерах, система надає прозорий інструмент моніторингу: клієнт у будь-який момент може зайти у свій особистий кабінет і побачити реальний статус виконання свого запиту («Очікує обробки», «В роботі», «Виконано»).

Отже, обґрунтування актуальності розробки полягає в тому, що створення спеціалізованої ІКС дозволяє підвищити швидкість обробки звернень, гарантувати високий рівень безпеки даних, усунути ризики втрати інформації та забезпечити масштабованість бізнес-процесів сервісної компанії без залучення надмірних людських ресурсів.

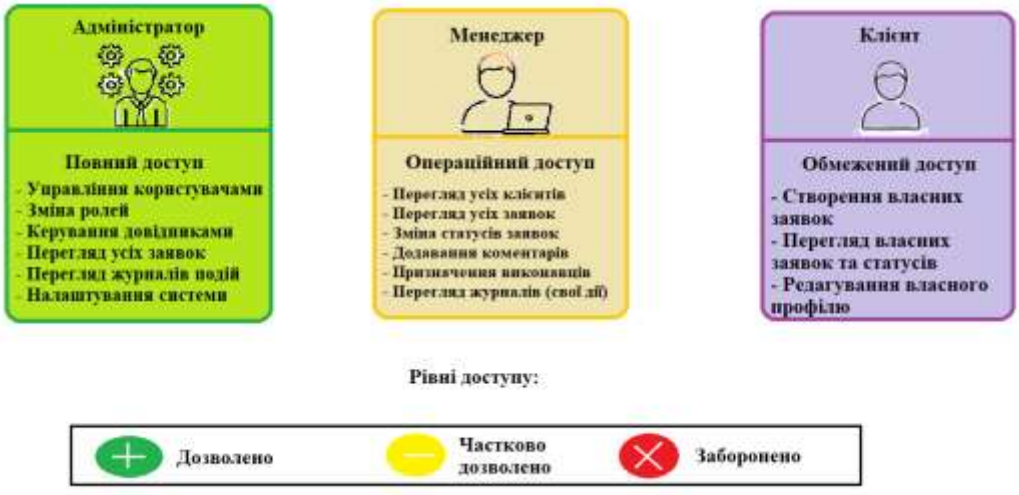

Рисунок 1.2 – Role-Based Access Control

1.3 Огляд та порівняльний аналіз існуючих аналогів програмних рішень

Перед розробкою власної інформаційно-комунікаційної системи необхідно провести детальне дослідження сучасного ринку програмного забезпечення, що застосовується для обліку клієнтів та обробки заявок. Це дозволить уникнути повторного створення вже існуючих архітектурних рішень («вигадування велосипеда»), чітко визначити переваги та недоліки наявних засобів, а також обґрунтувати розробку унікального продукту для обраної ніші.

На сьогодні сервісні IT-компанії та підприємства малого бізнесу використовують широкий спектр інструментів: від найпростіших локальних офісних програм до складних комерційних хмарних систем. Проведемо аналіз трьох найбільш розповсюджених категорій рішень:

1. Локальні електронні таблиці (на прикладі Microsoft Excel). Найбільш доступний та простий у первинному розгортанні інструмент. Він не потребує витрат на розробку чи закупівлю серверних потужностей. Проте Excel абсолютно не пристосований для одночасної багатокористувацької роботи, не має інструментів динамічного розмежування прав доступу і створює високі ризики повної втрати або витоку конфіденційних даних клієнтів [6].

2. Хмарні табличні сервіси (на прикладі Google Sheets). Забезпечують можливість спільного доступу до даних через Інтернет з будь-якої точки світу. Вони частково вирішують проблему мобільності менеджерів. Разом із тим, безпека та розмежування прав у таких сервісах є вкрай обмеженими [7]: неможливо приховати окремі поля чи записи (наприклад, описи критичних вразливостей систем клієнта) від певних категорій користувачів, не заблокувавши доступ до файлу повністю.

3. Універсальні комерційні CRM-системи (на прикладі Bitrix24). Мають надзвичайно широкий функціонал, що охоплює автоматизацію відділу продажів, маркетинг, лідогенерацію та наскрізну аналітику. Головним недоліком таких систем для невеликих сервісних IT-компаній є їхня надмірна складність,

перевантаженість інтерфейсу зайвими функціями, висока вартість ліцензій та складність адаптації [8] під специфіку надання саме IT-консультацій з кібербезпеки.

Для наочного зіставлення ключових характеристик та визначення «незаповненої ніші», яку має зайняти проєктувальний програмний продукт, побудовано порівняльну таблицю 1.1.

Таблиця 1.1 — Порівняльна характеристика існуючих рішень та проєктувальної ІКС

Назва рішення	**Тип системи**	**Сильні сторони (Переваги)**	**Слабкі сторони (Недоліки)**
Microsoft Excel	Локальний	Простота використання, відсутність витрат на розробку.	Відсутність багатокористувацької роботи, ризик втрати даних.
Google Sheets	Хмарний	Доступ онлайн з будь-якої точки, можливість спільної роботи.	Обмежені можливості безпеки та розмежування прав доступу.
CRM (напр. Bitrix24)	Веб-застосунок	Дуже широкий функціонал, автоматизація продажів.	Надмірна складність для малого бізнесу, висока ціна.
Проєктована ІКС	Веб-застосунок	Повна адаптація під потреби компанії, фокус на IT-консультаціях.	Потребує витрат часу на розробку та впровадження.

Проведений інженерний аналіз показує, що готові ринкові системи або занадто прості й незахищені (що загрожує конфіденційності клієнтів), або занадто громіздкі та дорогі (що перевантажує роботу операційного менеджера). Проєктована інформаційно-комунікаційна система покликана ефективно заповнити цю нішу завдяки трьом факторам:

- Вузька спеціалізація: орієнтація суто на облік заявок на IT-послуги та консультації з кібербезпеки;
- Гнучкий контроль: чітке розмежування прав за допомогою рольової моделі (Адміністратор, Менеджер, Клієнт) гарантує безпеку даних без ускладнення інтерфейсу;
- Кросплатформованість та веб-доступність: система працює безпосередньо через веб-браузер і не потребує встановлення додаткового клієнтського ПЗ на пристрої.

1.4 Постановка задачі системи

На основі детального аналізу предметної області, процесів взаємодії з клієнтами в сервісних IT-компаніях (підрозділ 1.1), обґрунтування інженерної необхідності автоматизації (підрозділ 1.2) та критичного огляду наявних програмних аналогів (підрозділ 1.3), постає задача розробки спеціалізованого програмного забезпечення.

Метою дипломної роботи є проєктування та програмна реалізація інформаційно-комунікаційної системи у вигляді кросплатформового веб-застосунку для автоматизації процесів обліку клієнтів та оперативного управління процесом надання консультаційних послуг з кібербезпеки.

Для досягнення поставленої мети в межах даної інженерної роботи необхідно вирішити такі завдання**:**

1. Виконати детальне функціональне та об'єктно-орієнтоване проєктування архітектури системи із побудовою відповідних UML-діаграм.

2. Спроєктувати схему та логічну структуру реляційної бази даних для надійного збереження профілів користувачів та журналів заявок.

3. Програмно реалізувати бекенд-частину веб-застосунку для забезпечення бізнес-логіки системи та обробки HTTP-запитів.

4. Розробити адаптивний користувацький веб-інтерфейс (фронтенд) для зручної взаємодії менеджерів та клієнтів із системою через браузер.

5. Реалізувати надійний механізм аутентифікації та рольову модель розмежування прав доступу користувачів до інформаційних ресурсів.

6. Провести комплексне функціональне тестування розробленого програмного продукту, перевірити коректність обробки помилок та валідації даних.

Об'єктом дослідження є процеси реєстрації, операційного обліку, моніторингу та зміни статусів заявок користувачів на отримання ІТ-послуг у сервісних компаніях.

Предметом дослідження є архітектурні підходи, моделі організації баз даних, методи розмежування прав доступу та програмні засоби створення веб-орієнтованих інформаційно-комунікаційних систем.

Для забезпечення чітких меж проєктування та відокремлення прикладного інженерного завдання від суміжних областей, встановлюються такі технічні обмеження та умови функціонування системи:

- Концептуальні обмеження: розробляється система виключно операційного обліку та комунікації. Система не є фінансовим чи бухгалтерським інструментом і не містить модулів онлайн-еквайрингу, виписки рахунків чи розрахунку заробітної плати персоналу.
- Специфіка кібербезпеки: роль кібербезпеки в межах цієї роботи є виключно *консультативною*. Проєкт спрямований на автоматизацію обліку заявок (наприклад, на проведення аудиту чи консультації) і не передбачає розробки низькорівневих сканерів вразливостей, міжмережевих екранів (файрволів) чи іншого спеціалізованого захисного ПЗ.

- Архітектурні обмеження: система будується за клієнт-серверною архітектурою. Доступ користувачів (адміністраторів, менеджерів, клієнтів) до функціональних можливостей здійснюється виключно через сучасні веб-браузери (Google Chrome, Mozilla Firefox, Microsoft Edge тощо) без необхідності встановлення додаткових клієнтських програм на пристрої.

РОЗДІЛ 2. АНАЛІЗ ІНЖЕНЕРНИХ МЕТОДІВ, АРХІТЕКТУРНИХ ПІДХОДІВ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

У цьому розділі наведено інженерний аналіз та теоретичне обґрунтування вибору архітектурних рішень, методів організації даних і технологічного стеку для розробки інформаційно-комунікаційної системи. Досліджено особливості розділення бізнес-логіки та інтерфейсу користувача за допомогою шаблонів проєктування, розглянуто моделі безпеки та критерії вибору інструментальних засобів бекенд- і фронтенд-розробки з метою створення масштабованого та захищеного веб-застосунку.

2.1 Дослідження архітектурних підходів до побудови веб-застосунків

Ефективність, гнучкість та підтримка сучасних інформаційно-комунікаційних систем значною мірою залежать від архітектурного шаблону (патерну), покладеного в основу їхнього проєктування. Для веб-орієнтованих застосунків де-факто стандартом в інженерії програмного забезпечення є концепція розділення відповідальності (Separation of Concerns) [9]. Вона передбачає ізоляцію логіки збереження даних від логіки обробки запитів та безпосереднього відображення інтерфейсу користувача.

Класичним втіленням цього принципу є архітектурний шаблон MVC (Model-View-Controller, Модель-Висвітлення-Контролер) трьох автономних компонентів [10]. Його структура базується на взаємодії трьох автономних компонентів:

- Model (Модель) — відповідає за рівень даних та бізнес-логіку застосунку. Вона взаємодіє із базою даних, здійснює вибірку, валідацію, збереження та модифікацію об'єктів (у нашому випадку — карток клієнтів та заявок).

- View (Висвітлення / Представлення) — відповідає за формування графічного інтерфейсу користувача (UI) та візуалізацію даних, отриманих від моделі.
- Controller (Контролер) — виступає посередником між Моделлю та Висвітленням. Він приймає вхідні HTTP-запити від користувача через браузер, інтерпретує їх, викликає відповідні методи моделі для обробки даних і передає результат у висвітлення для генерації відповіді.

У процесі розвитку сучасних веб-фреймворків, зокрема Python-фреймворку Django, класичний патерн MVC зазнав певної модифікації та трансформувався в архітектурну модель MVT [11] (Model-View-Template, Модель-Висвітлення-Шаблон). Незважаючи на зміну назв компонентів, концептуальна суть розділення логіки залишається незмінною, проте обов'язки між елементами розподіляються дещо інакше:

1. Model (Модель) в архітектурі MVT повністю еквівалентна моделі в MVC. Вона описує структуру даних за допомогою класів мови програмування і керує зв'язками з реляційною базою даних через механізм об'єктно-реляційного відображення (ORM).

2. View (Висвітлення) у системі MVT виконує роль, яка в класичному MVC покладалася на *Контролер*. Саме цей компонент містить функції або класи обробки запитів. Він приймає HTTP-запити, реалізує логіку взаємодії з моделями (наприклад, фільтрацію заявок клієнта за їхнім статусом) і вирішує, який саме HTML-шаблон та з якими контекстними даними потрібно повернути користувачеві.

3. Template (Шаблон) замінює компонент View із класичного MVC. Це текстовий файл (найчастіше HTML-сторінка), що містить статичну розмітку та спеціальні теги мови шаблонізатора. Шаблон відповідає виключно за те, як дані будуть відображені у веб-браузері. Він динамічно рендериться на стороні сервера, заповнюючись даними, які йому передав компонент View.

Сам фреймворк у такій схемі бере на себе функції контролера [11], зокрема автоматичну маршрутизацію URL-адрес (URL routing). Коли користувач

переходить за певним посиланням, вбудований контролер фреймворку аналізує запит і перенаправляє його до відповідного Висвітлення (View).

Дослідження показує, що використання підходу MVT для проєктування ІКС обліку клієнтів має суттєві інженерні переваги (рис. 2.1):

- Model описує структуру даних через класи Python та взаємодіє з БД через ORM..
- View містить бізнес-логіку, приймає HTTP-запити, фільтрує дані відповідно до прав користувача та передає контекст у шаблон.
- Template відповідає за HTML-розмітку та відображення даних у браузері. Маршрутизацію (URL Routing) фреймворк бере на себе, виступаючи в ролі загального контролера.

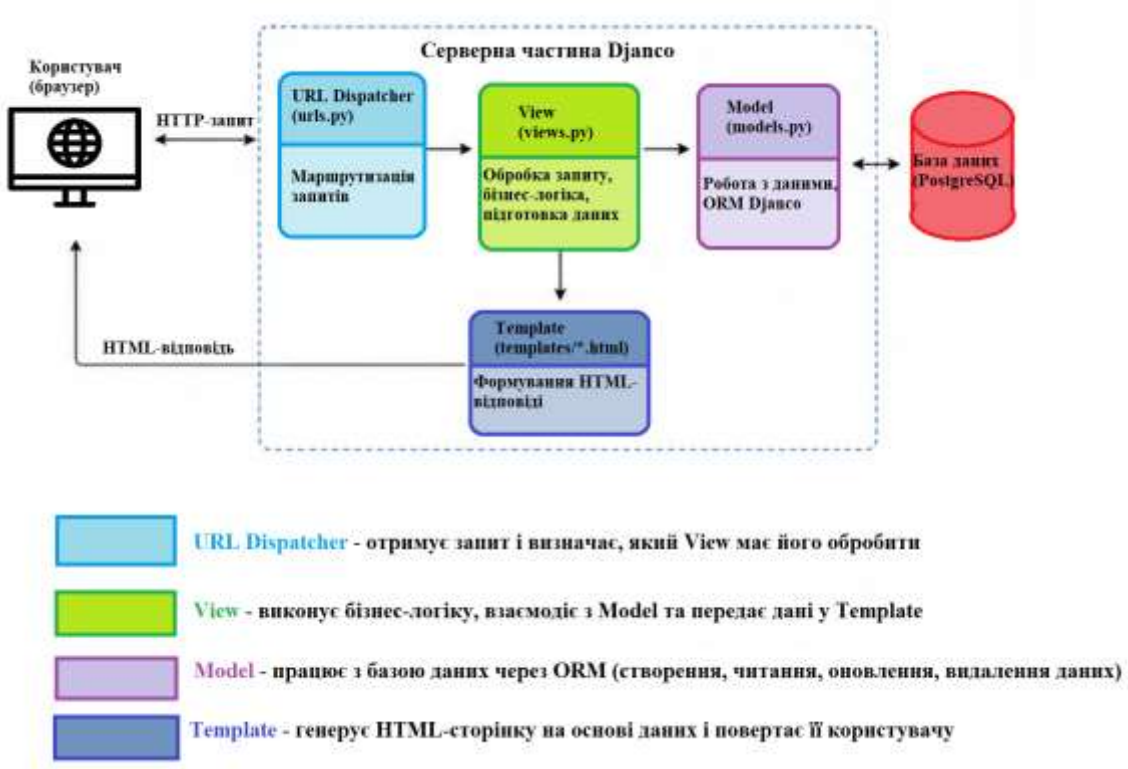

Рисунок 2.1 – Архітектура MVC/MVT у Django

Таким чином, архітектурний шаблон MVT є найбільш доцільним інженерним підходом для побудови розроблюваної інформаційно-комунікаційної системи, оскільки він забезпечує високу швидкість розробки, надійну ізоляцію

логіки безпеки від інтерфейсу та простоту подальшого масштабування програмного продукту [10].

2.2 Моделі організації доступу до даних та безпеки взаємодії користувачів

Проєкт інформаційно-комунікаційної системи, що забезпечує облік заявок на консультаційні послуги з кібербезпеки, вимагає особливого підходу до проектування контуру безпеки. Оскільки дані, які вносять клієнти (інформація про поточний стан захищеності, виявлені інциденти чи архітектурні особливості їхніх IT-ресурсів), мають високий ступінь чутливості, архітектура системи повинна гарантувати сувору конфіденційність та цілісність інформації на всіх етапах її життєвого циклу.

Основним методом забезпечення безпеки на рівні логіки застосунку є впровадження рольової моделі доступу (Role-Based Access Control, RBAC). Концепція RBAC передбачає межі його повноважень [12], що права на виконання певних операцій (читання, запис, модифікація, видалення) у системі прив'язуються не до конкретних облікових записів користувачів, а до абстрактних ролей. Користувачеві, у свою чергу, призначається одна або декілька таких ролей, що автоматично визначає межі його повноважень.

Для розроблюваної ІКС обґрунтовано виділення трьох базових ролей із суворим розмежуванням прав доступу до об'єктів бази даних:

- Клієнт (обмежений рівень доступу) — має повноваження на реєстрацію, автентифікацію, створення та редагування власного профілю, а також ініціацію нових заявок на консультації. Головне інженерне обмеження на рівні запитів (Query-level filtering) полягає в тому, що клієнт може переглядати та відстежувати статуси виключно своїх власних заявок. Доступ до чужих профілів чи загального списку звернень повністю блокується на рівні бізнес-логики бекенду.

- Менеджер (операційний рівень доступу) — володіє ширшими повноваженнями, що дозволяють здійснювати наскрізний перегляд бази даних клієнтів та керувати журналом усіх вхідних заявок. Менеджер має право змінювати життєві статуси запитів (наприклад, переводити у стан «В роботі» або «Завершено»), призначати відповідальних фахівців та додавати технічні коментарі до консультацій. Проте операційний менеджер позбавлений прав на деструктивні дії (видалення користувачів) та зміну глобальних налаштувань системи.
- Адміністратор (повний рівень доступу) — має повний рівень доступу [12]. Також має доступ до керування обліковими записами всіх користувачів (створення, блокування, зміна ролей), редагування системних довідників (перелік та вартість консультаційних послуг) та аудиту безпеки логів системи.

З інженерної точки зору безпека взаємодії користувачів із системою реалізується у кілька етапів, утворюючи захищений стек:

1. Ідентифікація та автентифікація. Процес перевірки справжності користувача здійснюється за допомогою пари «логін (або e-mail) — пароль». Зберігання паролів у відкритому вигляді в базі даних категорично заборонено. Для забезпечення криптографічного захисту застосовується метод одностороннього хешування за допомогою стійких алгоритмів (наприклад, PBKDF2 або bcrypt) із обов'язковим додаванням випадкової криптографічної солі (salting) для захисту від атак за допомогою райдужних таблиць [13].

2. Авторизація та контроль сесій. Після успішного входу система ініціює механізм контролю сесій на основі захищених HTTP-кук (Session-based authentication) або токенів (Token-based authentication). Кожен вхідний HTTP-запит перевіряється проміжним програмним забезпеченням (Middleware) на бекенді на предмет наявності активної сесії та відповідності ролі користувача запитуваному URL-маршруту. Спроби неавторизованого доступу автоматично відхиляються з поверненням коду помилки HTTP 403 Forbidden або перенаправленням на сторінку автентифікації.

3. Захист від поширених веб-уразливостей. Для гарантування безпеки взаємодії на рівні передачі даних обов'язковим є використання шифрування трафіку за протоколом HTTPS [14]. Крім того, на рівні обробки форм та взаємодії з базою даних інформаційна система має містити вбудований захист від таких критичних загроз, як:

- SQL-ін'єкції (SQLi): нейтралізуються за рахунок використання ORM-інструментів, які автоматично параметризують усі SQL-запити [15] до бази даних;
- Міжсайтова підробка запиту (CSRF): блокується шляхом генерації та обов'язкової валідації унікальних CSRF-токенів [16] для кожної POST-форми;
- Міжсайтовий скриптинг (XSS): усувається завдяки автоматичному екрануванню (escaping) [17] вхідних текстових даних у шаблонах інтерфейсу.

Впровадження описаного комплексу методів організації доступу та безпеки дозволяє створити надійне ізольоване середовище, яке відповідає специфіці надання консультацій з кібербезпеки та унеможливлює несанкціоновану модифікацію чи витік інформації.

2.3 Обґрунтування вибору технологічного стеку для бекенд-робки

Вибір інструментальних засобів для реалізації серверної логіки (Backend) є одним із ключових етапів проєктування інформаційно-комунікаційної системи. Від архітектурних особливостей мови програмування та обраного веб-фреймворку безпосередньо залежать такі характеристики програмного продукту, як швидкість розробки, рівень безпеки даних, простота супроводу коду та масштабованість системи.

Для розробки бекенд-частини ІКС обліку клієнтів було проведено порівняльний інженерний аналіз трьох найбільш популярних високорівневих мов програмування, що використовуються у сучасній веб-індустрії:

1. PHP (фреймворк Laravel). Традиційна та найбільш розповсюджена мова для створення веб-застосунків. Laravel має потужні вбудовані інструменти для

роботи з базами даних (Eloquent ORM) та готові модулі автентифікації. Проте PHP історично має репутацію мови з менш суворими стандартами типізації, що за відсутності належного контролю може призводити до появи архітектурних помилок у великих проєктах [18].

2. JavaScript / TypeScript (платформа Node.js). Забезпечує високу швидкість обробки великої кількості одночасних запитів завдяки асинхронній, подійно-орієнтованій моделі введення-виведення (Non-blocking I/O). Головною перевагою є можливість використання однієї мови як на фронтенді, так і на бекенді. Водночас екосистема Node.js швидко змінюється, що ускладнює довготривалу підтримку (LTS) та вимагає ручного налаштування багатьох компонентів безпеки [19].

3. Python (фреймворк Django). Високорівнева мова програмування з чітким, читабельним синтаксисом, яка підтримує концепцію швидкої розробки (Rapid Development). Веб-фреймворк Django побудований за принципом «все включено» (Batteries Included), що означає наявність готових інструментів для вирішення більшості стандартних інженерних задач (адміністративна панель, ORM, валідація форм, механізми міграцій) [20].

У результаті зіставлення критичних вимог до проєктувальної ІКС — а саме необхідності забезпечення максимального рівня безпеки даних клієнтів (через специфіку консультацій з кібербезпеки) та високої швидкості розробки в межах дипломного проєкту — для реалізації системи було обрано мову Python та фреймворк Django.

Даний інженерний вибір обґрунтований такими ключовими перевагами стеку Python/Django:

- Вбудований контур безпеки (Security Out-of-the-Box). Django проєктувався з фокусом на запобігання поширеним помилкам розробників. Він містить автоматичні вбудовані механізми захисту від SQL-ін'єкцій, міжсайтового скриптингу (XSS) та CSRF-атак на рівні ядра, що критично важливо для системи, яка оперує даними про IT-вразливості клієнтів.

- Потужний інструмент ORM (Object-Relational Mapping). Django ORM дозволяє описувати схему бази даних та взаємодіяти з нею виключно через класи та об'єкти Python, повністю абстрагуючись від написання сирих SQL-запитів. Це мінімізує помилки сумісності та прискорює проєктування структури даних.
- Автоматична адміністративна панель. Фреймворк автоматично генерує функціональний веб-інтерфейс для моделей даних. Це дозволяє без додаткових витрат часу реалізувати повноцінне робоче місце для ролі Адміністратора, забезпечуючи можливість зручного керування користувачами та довідниками системних послуг з перших етапів розгортання проєкту.
- Розвинена екосистема та підтримка модульності. Наявність великої кількості перевірених сторонніх бібліотек дозволяє легко інтегрувати додаткові інструменти (наприклад, системи логування, аналітичні модулі чи засоби автентифікації), що підвищує загальну гнучкість системи.

Таким чином, використання технологічного стеку на базі Python та Django є інженерно виправданим і доцільним рішенням, оскільки воно дозволяє створити надійну, безпечну та структуровану серверну частину веб-застосунку з мінімальними часовими витратами на рутинне кодування базової інфраструктури ПЗ.

2.4 Обґрунтування вибору системи управління базами даних та фронтенд-засобів

Окрім проєктування серверної логіки, важливим інженерним завданням є вибір системи управління базами даних (СУБД) та інструментів побудови інтерфейсу користувача (Front-end). Ці компоненти забезпечують фізичне збереження інформаційних масивів і безпосередню взаємодію користувачів із функціоналом ІКС.

Для вибору СУБД було проведено порівняльний аналіз двох концептуальних підходів [21]: реляційного (SQL) та нереляційного (NoSQL).

Системи класу NoSQL (наприклад, MongoDB) орієнтовані на збереження неструктурованих даних у вигляді документів, що забезпечує високу гнучкість, але ускладнює контроль цілісності при складних зв'язках. Оскільки розроблювана інформаційно-комунікаційна система оперує суворо структурованими даними (користувачі, ролі, заявки, статуси), які мають чіткі взаємозв'язки типу «один-до-багатьох» (наприклад, один клієнт може мати багато заявок), обґрунтованим є використання реляційної СУБД [21].

Серед реляційних рішень було розглянуто СУБД MySQL та PostgreSQL. Обидві системи є безкоштовними (Open Source) та мають високу швидкодію. Проте для реалізації проєкту було обрано СУБД PostgreSQL (або сумісну MySQL, інтегровану через Django ORM), оскільки вона забезпечує повну відповідність критеріям ACID [22] (атомарність, узгодженість, ізольованість, довговічність), підтримує складні транзакції, має розширені механізми безпеки та ідеально інтегрується з фреймворком Django на рівні драйверів бази даних.

Для розробки клієнтської частини (Front-end) веб-застосунку, що функціонує безпосередньо у браузері користувача, було обрано класичний стек веб-технологій та допоміжні бібліотеки:

- HTML5 (HyperText Markup Language) — використовується для проєктування семантичної структури та каркаса веб-сторінок системи (форм реєстрації, таблиць обліку заявок, профілів користувачів);
- CSS3 (Cascading Style Sheets) — застосовується для стилізації компонентів інтерфейсу, забезпечення адаптивності (Responsive Design) для коректного відображення системи на мобільних пристроях та ПК, а також побудови зручного UI/UX;
- Фреймворк Bootstrap — обраний як допоміжний інструмент для прискорення верстки за рахунок використання готових адаптивних компонентів [23] (навігаційні панелі, модальні вікна, кнопки, форми), що гармонізує інтерфейс без написання великих обсягів сирого CSS-коду;
- JavaScript — мова програмування фронтенду, що використовується для забезпечення динамічної взаємодії з користувачем, валідації введених у

форми даних на стороні клієнта (до відправки на сервер) та реалізації асинхронних запитів (AJAX) [24] для оновлення статусів заявок без повного перезавантаження сторінок браузера.

Описаний стек Front-end засобів дозволяє створити легкий, інтуїтивно зрозумілий інтерфейс, який працює на будь-яких пристроях і забезпечує комфортну роботу менеджерів та клієнтів.

РОЗДІЛ 3. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНОЇ СИСТЕМИ

У цьому розділі виконується повний комплекс інженерно-проєктних робіт зі створення інформаційно-комунікаційної системи. На основі проведеного раніше аналізу предметної області та обраних архітектурних підходів сформульовано детальні функціональні та нефункціональні вимоги до програмного продукту. Наведено специфікацію ролей користувачів, спроєктовано матрицю прав доступу та визначено архітектурні межі взаємодії між клієнтською та серверною частинами веб-застосунку.

3.1 Розробка вимог до системи та визначення ролей користувачів

Проєктні інженерні рішення щодо створення будь-якого програмного забезпечення базуються на чіткому формуванні технічних вимог. Відповідно до методологій інженерії програмного забезпечення, вимоги до проєктувальної ІКС обліку клієнтів розподіляються на дві основні категорії: функціональні (Functional Requirements) та нефункціональні (Non-functional Requirements) [25].

Функціональні вимоги визначають безпосередні завдання, які система повинна виконувати для задоволення потреб користувачів. До ключових функціональних вимог належать:

- забезпечення можливості самостійної онлайн-реєстрації клієнтів та створення персонального кабінету;
- автентифікація користувачів у системі за допомогою унікальних ідентифікаторів та захищених паролів;
- надання інструментарію для формування електронних заявок на отримання консультаційних послуг з кібербезпеки із зазначенням типу послуги та опису проблеми;
- забезпечення динамічної зміни статусів обробки заявок операційним менеджером;

- автоматичне логування дій та централізоване збереження історії всіх звернень клієнтів у базі даних;
- надання адміністратору повного інтерфейсу для моніторингу користувачів та редагування довідників системи.

Нефункціональні вимоги визначають якісні характеристики системи, умови її експлуатації та архітектурні обмеження:

- Безпека та конфіденційність: система повинна забезпечувати надійне шифрування паролів на рівні БД, захист від веб-атак (SQLi, CSRF, XSS) та повну ізоляцію даних між різними обліковими записами клієнтів.
- Кросплатформованість та доступність: стабільне функціонування інтерфейсу у всіх поширених веб-браузерах на операційних системах Windows, Linux, macOS, Android та iOS.
- Ергономічність та швидкодія: час відгуку системи на стандартні HTTP-запити користувача не повинен перевищувати 1.5–2 секунд за нормальних умов мережевого з’єднання, а інтерфейс має бути інтуїтивно зрозумілим.

Будь-яка інформаційно-комунікаційна система існує для користувачів, тому архітектура проєкту будується навколо концепції управління профілями. На етапі аналізу технологічного стеку (розділ 2) було обґрунтовано впровадження рольової моделі доступу (RBAC). Для деталізації функціональних можливостей кожної ролі розроблено сувору специфікацію прав.

Для наочного представлення та інженерної фіксації меж повноважень у системі розроблено матрицю прав доступу, яку наведено в табл. 3.1.

Таблиця 3.1 — Матриця прав доступу користувачів до функціональних модулів ІКС

Функціональний модуль системи	**Клієнт**	**Менеджер**	**Адміністратор**
Реєстрація та редагування власного профілю	Дозволено	Дозволено	Дозволено
Створення нової заявки на консультацію	Дозволено	Заборонено	Заборонено
Перегляд власних заявок та їхніх статусів	Дозволено	Дозволено	Дозволено
Перегляд повного журналу заявок усіх клієнтів	Заборонено	Дозволено	Дозволено
Модифікація статусів заявок та призначення виконавців	Заборонено	Дозволено	Дозволено
Керування обліковими записами (блокування, зміна ролей)	Заборонено	Заборонено	Дозволено
Редагування системних довідників та конфігурацій	Заборонено	Заборонено	Дозволено

Логіка взаємодії користувачів із системою будується на принципі мінімальних привілеїв [26]. Клієнтська взаємодія ізольована на рівні ідентифікатора сесії, операційні менеджери використовують спільну чергу обробки даних, а адміністратор керує метаданими через захищений системний контур. Логічну модель взаємодії акторів формалізовано на UML Use Case діаграмі (рис. 3.1).

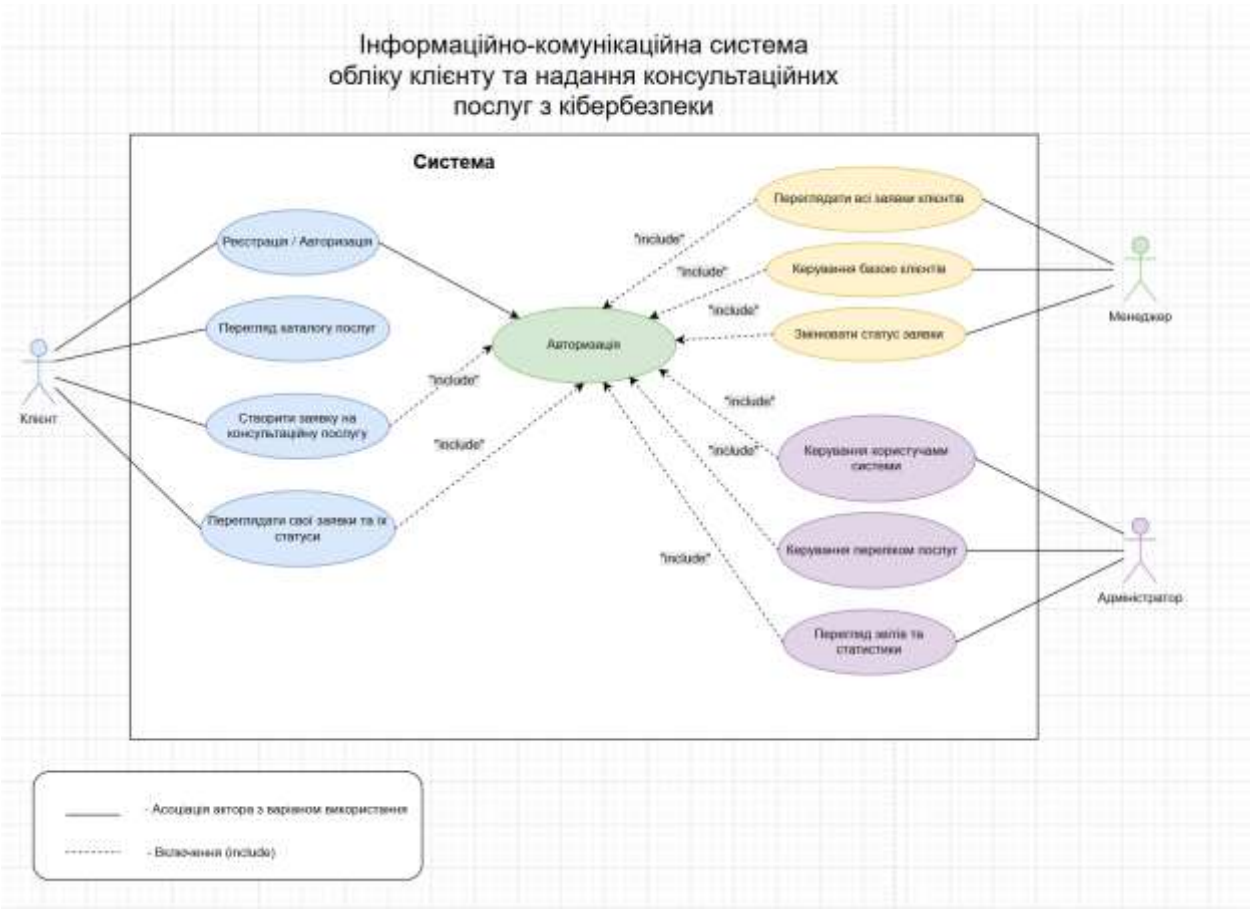

Рисунок 3.1 – Діаграма прецедентів (Use Case Diagram)

3.2 Функціональна структура інформаційно-комунікаційної системи

Функціональна структура інформаційно-комунікаційної системи визначає набір програмних модулів, їх взаємозв’язок та логіку обробки інформаційних потоків між користувачами та серверною частиною веб-застосунку. Правильне розділення системи на окремі функціональні компоненти забезпечує

масштабованість програмного забезпечення, спрощує подальшу підтримку коду та підвищує загальну надійність системи [27].

Розроблювана інформаційно-комунікаційна система обліку клієнтів та надання консультаційних послуг з кібербезпеки будується за модульним принципом. Кожен функціональний модуль виконує окрему групу завдань та взаємодіє з іншими компонентами через механізми серверної логіки та реляційної бази даних.

До складу системи входять такі основні функціональні модулі:

- модуль автентифікації та авторизації користувачів;
- модуль управління профілями клієнтів;
- модуль обробки заявок на IT-послуги;
- модуль адміністрування;
- модуль журналювання подій;
- модуль взаємодії з базою даних;
- модуль веб-інтерфейсу користувача.

Модуль автентифікації та авторизації

Одним із найважливіших компонентів інформаційної системи є модуль автентифікації та авторизації користувачів. Його головним призначенням є перевірка достовірності облікових даних користувача та визначення рівня доступу до функціональних можливостей системи.

Автентифікація користувачів здійснюється за допомогою унікального логіна або електронної пошти та пароля. Для забезпечення безпеки паролі користувачів не зберігаються у відкритому вигляді [28]. Перед записом у базу даних пароль проходить процедуру криптографічного хешування із використанням сучасних алгоритмів захисту.

Після успішного входу система створює захищену користувацьку сесію та визначає роль користувача [29]. На основі ролі формується набір доступних функцій і сторінок системи.

У системі реалізовано три основні ролі користувачів:

1. Адміністратор.
2. Менеджер.
3. Клієнт.

Кожна роль має власний набір дозволених операцій та обмежень доступу.

Модуль управління профілями клієнтів

Модуль управління профілями забезпечує централізоване збереження інформації про клієнтів сервісної компанії. Основним призначенням модуля є автоматизація процесів реєстрації, редагування та перегляду персональних даних користувачів.

У профілі клієнта зберігаються такі дані:

- прізвище та ім’я;
- контактний номер телефону;
- адреса електронної пошти;
- дата реєстрації;
- історія звернень;
- статус активності облікового запису.

Менеджер системи має можливість переглядати список клієнтів, здійснювати пошук за різними параметрами та редагувати інформацію у разі необхідності. Клієнт, у свою чергу, може змінювати лише власні персональні дані.

Зберігання інформації у централізованій базі даних дозволяє уникнути дублювання записів та забезпечує швидкий доступ до історії взаємодії з клієнтами.

Модуль обробки заявок на IT-послуги

Модуль обробки заявок є основним функціональним компонентом системи. Саме через нього здійснюється взаємодія між клієнтом та сервісною IT-компанією.

Після авторизації клієнт отримує можливість створити нову заявку на консультаційні послуги з кібербезпеки. Під час створення заявки користувач заповнює спеціальну форму, у якій вказує:

- тип послуги;
- опис проблеми;
- пріоритетність звернення;
- додаткову інформацію щодо інциденту.

Після надсилання заявки система автоматично присвоює їй унікальний ідентифікатор та статус «Очікує обробки». Усі створені заявки потрапляють до журналу менеджера.

Менеджер має можливість:

- переглядати список усіх заявок;
- змінювати статус заявки;
- додавати коментарі;
- призначати відповідального виконавця;
- закривати заявку після завершення роботи.

Для забезпечення прозорості обслуговування клієнт може в режимі реального часу переглядати поточний статус власної заявки через веб-інтерфейс системи.

Модуль адміністрування

Модуль адміністрування призначений для управління глобальними параметрами системи та контролю користувачів.

Адміністратор системи володіє найвищим рівнем доступу та має можливість:

- створювати нові облікові записи;
- змінювати ролі користувачів;
- блокувати користувачів;
- редагувати довідники послуг;
- здійснювати аудит подій системи;
- переглядати журнали активності.

Для реалізації функцій адміністрування використовується вбудована адміністративна панель Django [30], яка автоматично генерує веб-інтерфейс для управління моделями бази даних.

Модуль журналювання подій

Для підвищення рівня безпеки та забезпечення контролю за діями користувачів у системі реалізовано модуль журналювання подій.

Модуль автоматично фіксує:

- спроби входу в систему;
- створення нових заявок;
- зміну статусів;
- редагування даних;
- адміністративні операції;
- помилки авторизації.

Журнали подій дозволяють здійснювати аудит безпеки та аналізувати потенційні порушення доступу до інформаційних ресурсів.

Модуль взаємодії з базою даних

Для централізованого збереження інформації використовується реляційна система управління базами даних PostgreSQL.

Взаємодія серверної частини з базою даних здійснюється через ORM-механізм фреймворку Django [31]. Такий підхід дозволяє працювати з таблицями бази даних у вигляді об’єктів Python без необхідності написання великої кількості SQL-запитів.

Основними перевагами використання ORM є [31]:

- зменшення кількості програмних помилок;
- автоматичне формування SQL-запитів;
- підвищення безпеки доступу до даних;
- підтримка міграцій структури бази даних.

Модуль веб-інтерфейсу користувача

Веб-інтерфейс системи забезпечує взаємодію користувачів із функціональними можливостями програмного продукту через браузер.

Інтерфейс реалізовано за допомогою технологій HTML5, CSS3, Bootstrap та JavaScript [23]. Використання адаптивної верстки дозволяє коректно відображати сторінки системи як на персональних комп’ютерах, так і на мобільних пристроях.

Основними елементами користувацького інтерфейсу є:

- форма входу;
- сторінка реєстрації;
- особистий кабінет клієнта;
- таблиця заявок;
- панель адміністратора;
- форма створення заявки.

Для покращення взаємодії користувача із системою реалізовано механізми валідації форм та асинхронного оновлення даних [32] без повного перезавантаження сторінок.

Взаємодія функціональних компонентів системи

Узагальнена схема взаємодії компонентів інформаційно-комунікаційної системи базується на клієнт-серверній архітектурі.

Користувач взаємодіє із системою через веб-браузер. HTTP-запити передаються на серверну частину Django-застосунку, де відбувається обробка бізнес-логіки. Сервер здійснює взаємодію з базою даних PostgreSQL та повертає сформовану HTML-відповідь користувачеві.

Такий підхід забезпечує:

- централізоване збереження інформації;
- високу швидкість обробки даних;
- безпечну взаємодію між компонентами;
- масштабованість програмного забезпечення;
- підтримку багатокористувацького режиму роботи.

Отже, функціональна структура інформаційно-комунікаційної системи забезпечує чітке розділення відповідальності між програмними модулями та створює надійну основу для реалізації веб-застосунку обліку клієнтів і надання консультаційних послуг з кібербезпеки.

3.3 Проєктування бази даних інформаційно-комунікаційної системи

Одним із ключових етапів розробки інформаційно-комунікаційної системи є проєктування структури бази даних. Саме база даних забезпечує централізоване збереження інформації, підтримку цілісності даних, швидкий доступ до інформаційних ресурсів та стабільну роботу всієї системи.

Для реалізації інформаційно-комунікаційної системи обліку клієнтів та надання консультаційних послуг з кібербезпеки використовується реляційна база даних PostgreSQL. Вибір реляційного підходу обґрунтовується необхідністю забезпечення чітких зв’язків між сутностями системи [33], підтримкою транзакцій та високим рівнем надійності збереження інформації.

Основні вимоги до бази даних

Проєктована база даних повинна забезпечувати:

- централізоване збереження інформації;
- підтримку багатокористувацького режиму роботи;
- високу швидкість обробки запитів;
- захист від втрати та пошкодження інформації;
- підтримку механізмів резервного копіювання;
- підтримку цілісності та узгодженості даних;
- можливість масштабування системи.

У процесі проєктування бази даних було визначено основні сутності предметної області та взаємозв’язки між ними.

До основних сутностей системи належать:

- користувачі;
- ролі;
- клієнти;
- заявки;
- послуги;
- журнали подій.

ER-модель бази даних

ER-модель (Entity Relationship Model) використовується для графічного представлення структури бази даних [34] та зв’язків між сутностями (див. в рис. 3.2).

Основними сутностями бази даних є:

1. Users.
2. Roles.
3. Clients.
4. Requests.
5. Services.
6. Logs.

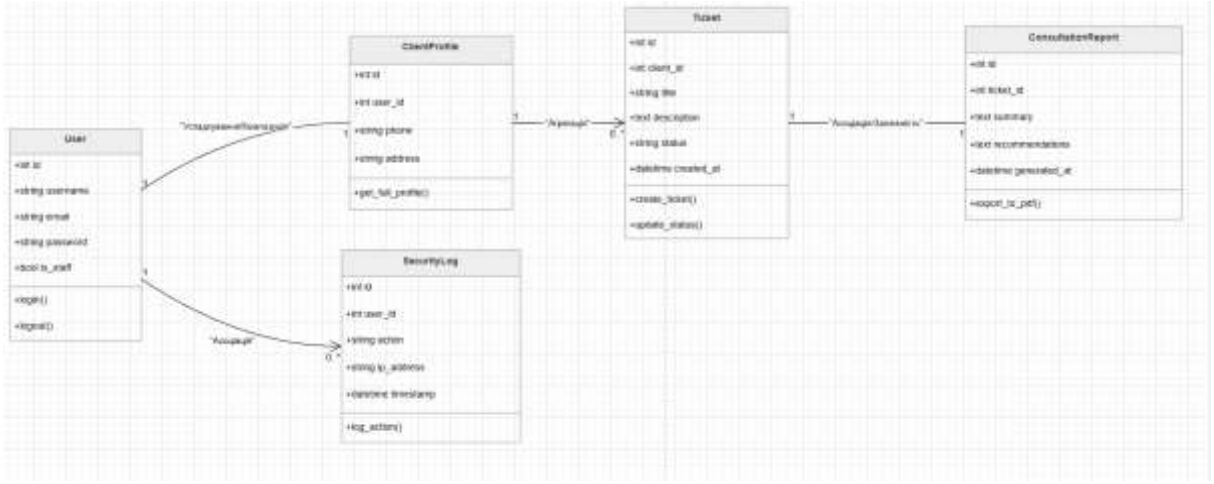

Рисунок 3.2 – Діаграма класів системи (Class Diagram)

Між таблицями реалізуються зв’язки типу:

- один-до-багатьох;
- багато-до-одного.

Наприклад:

- один клієнт може створити багато заявок;
- одна послуга може бути пов’язана з багатьма заявками;
- одна роль може бути призначена багатьом користувачам.

ER-діаграма дозволяє наочно представити логіку взаємодії між об’єктами системи та спрощує процес реалізації структури бази даних.

Таблиця Roles

Таблиця Roles призначена для збереження ролей користувачів системи.

Основними ролями є:

- Адміністратор;
- Менеджер;
- Клієнт.

Структура таблиці Roles:

Поле	**Тип даних**	**Опис**
id	INTEGER	Первинний ключ
role_name	VARCHAR(50)	Назва ролі
description	TEXT	Опис ролі

Первинним ключем таблиці є поле id.

Таблиця дозволяє реалізувати рольову модель доступу RBAC та централізовано управляти рівнями доступу користувачів.

Таблиця Users

Таблиця Users використовується для збереження інформації про користувачів системи.

Структура таблиці Users:

Поле	**Тип даних**	**Опис**
id	INTEGER	Первинний ключ
username	VARCHAR(100)	Логін користувача
email	VARCHAR(100)	Електронна пошта
password_hash	VARCHAR(255)	Хеш пароля
role_id	INTEGER	Зовнішній ключ
created_at	TIMESTAMP	Дата реєстрації
is_active	BOOLEAN	Статус активності

Поле role_id є зовнішнім ключем та пов’язує користувача з таблицею Roles.

Для забезпечення безпеки пароль користувача зберігається у вигляді криптографічного хешу.

Таблиця Clients

Таблиця Clients призначена для збереження персональної інформації клієнтів.

Структура таблиці Clients:

Поле	**Тип даних**	**Опис**
id	INTEGER	Первинний ключ
user_id	INTEGER	Зовнішній ключ
first_name	VARCHAR(100)	Ім’я
last_name	VARCHAR(100)	Прізвище
phone	VARCHAR(20)	Телефон
company_name	VARCHAR(150)	Назва компанії
registration_date	DATE	Дата реєстрації

Поле user_id забезпечує зв’язок між клієнтом та обліковим записом користувача.

Один користувач може мати лише один клієнтський профіль.

Таблиця Services

Таблиця Services містить інформацію про доступні консультаційні послуги.

Структура таблиці Services:

Поле	**Тип даних**	**Опис**
id	INTEGER	Первинний ключ
service_name	VARCHAR(150)	Назва послуги
description	TEXT	Опис послуги
price	DECIMAL(10,2)	Вартість
duration	INTEGER	Тривалість

У таблиці зберігаються всі типи послуг, які надає сервісна IT-компанія.

Таблиця Requests

Таблиця Requests є основною таблицею системи та використовується для збереження заявок клієнтів.

Структура таблиці Requests:

Поле	**Тип даних**	**Опис**
id	INTEGER	Первинний ключ
client_id	INTEGER	Зовнішній ключ
service_id	INTEGER	Зовнішній ключ
title	VARCHAR(200)	Назва заявки
description	TEXT	Опис проблеми
status	VARCHAR(50)	Статус заявки
created_at	TIMESTAMP	Дата створення
updated_at	TIMESTAMP	Дата оновлення

Поле client_id забезпечує зв’язок із таблицею Clients, а поле service_id — із таблицею Services.

Для кожної заявки система автоматично фіксує дату створення та останнього оновлення.

Основними статусами заявки є:

- Очікує обробки;
- У роботі;

- Виконано;
- Закрито.

Таблиця Logs

Таблиця Logs використовується для журналювання подій системи.

Структура таблиці Logs:

Поле	**Тип даних**	**Опис**
id	INTEGER	Первинний ключ
user_id	INTEGER	Зовнішній ключ
action	VARCHAR(255)	Тип дії
created_at	TIMESTAMP	Дата події
ip_address	VARCHAR(50)	IP-адреса

Журналювання дозволяє здійснювати аудит дій користувачів та підвищує загальний рівень безпеки інформаційної системи.

Нормалізація бази даних

Під час проєктування структури бази даних було застосовано принципи нормалізації для усунення дублювання інформації та забезпечення цілісності даних.

База даних приведена до третьої нормальної форми (3NF) [35].

Перша нормальна форма забезпечує:

- атомарність значень;
- відсутність повторюваних груп.

Друга нормальна форма усуває часткові залежності між атрибутами.

Третя нормальна форма забезпечує відсутність транзитивних залежностей між полями таблиць.

Використання нормалізації дозволяє:

- зменшити дублювання даних;
- спростити оновлення інформації;
- уникнути аномалій вставки та видалення;

- забезпечити узгодженість інформації.

Забезпечення цілісності даних

Для забезпечення цілісності інформації у базі даних використовуються:

- первинні ключі;
- зовнішні ключі;
- обмеження NOT NULL;
- обмеження UNIQUE;
- каскадні зв’язки;
- перевірки типів даних.

Зовнішні ключі дозволяють підтримувати коректність зв’язків між таблицями та запобігають появі неконсистентних записів.

Індексація таблиць

Для підвищення швидкодії системи у базі даних застосовуються індекси [33].

Індекси створюються для:

- ідентифікаторів користувачів;
- електронних адрес;
- статусів заявок;
- дат створення заявок.

Використання індексів значно прискорює виконання SQL-запитів та підвищує продуктивність системи при роботі з великими обсягами інформації.

Резервне копіювання бази даних

Одним із важливих елементів забезпечення надійності системи є резервне копіювання бази даних.

Резервне копіювання дозволяє:

- відновити інформацію після збою;
- захистити систему від втрати даних;
- забезпечити безперервність роботи.

Для PostgreSQL можуть використовуватися:

- автоматичні резервні копії;

- експорт SQL-дампів;
- хмарне резервування.

У результаті проєктування бази даних було сформовано реляційну структуру, яка забезпечує надійне збереження інформації про користувачів, клієнтів, заявки та консультаційні послуги.

Розроблена структура бази даних відповідає вимогам цілісності, безпеки та масштабованості, а також забезпечує ефективну взаємодію всіх функціональних компонентів інформаційно-комунікаційної системи (рис. 3.3).

Рисунок 3.3 – Діаграма послідовностей обробки заявки (Sequence Diagram)

3.4 UML-діаграми інформаційно-комунікаційної системи

Для якісного проєктування програмного забезпечення необхідним є використання спеціалізованих засобів моделювання структури та поведінки системи. Одним із найбільш поширених стандартів візуального моделювання програмних продуктів є UML (Unified Modeling Language) [36].

UML-діаграми дозволяють формалізувати архітектуру інформаційної системи, визначити взаємодію між компонентами, описати ролі користувачів та спростити процес реалізації програмного забезпечення.

Для проєктування інформаційно-комунікаційної системи обліку клієнтів та надання консультаційних послуг з кібербезпеки було використано такі UML-діаграми:

- Use Case Diagram;
- Class Diagram;
- Sequence Diagram.

Use Case Diagram

Use Case Diagram використовується для моделювання взаємодії користувачів із функціональними можливостями системи [36].

Основним призначенням діаграми варіантів використання є:

- визначення функцій системи;
- опис ролей користувачів;
- формування меж інформаційної системи;
- відображення взаємодії між користувачами та програмним продуктом.

У розроблюваній системі визначено три основні актори:

1. Клієнт.
2. Менеджер.
3. Адміністратор.

Клієнт взаємодіє із системою через особистий кабінет та виконує такі операції:

- реєстрація;
- авторизація;

- створення заявки;
- перегляд статусу заявки;
- редагування профілю.

Менеджер має ширший набір функцій:

- перегляд усіх заявок;
- зміна статусів;
- додавання коментарів;
- призначення виконавців;
- перегляд інформації про клієнтів.

Адміністратор системи виконує функції управління:

- створення користувачів;
- зміна ролей;
- блокування облікових записів;
- редагування довідників послуг;
- перегляд журналів подій.

Use Case Diagram дозволяє наочно продемонструвати межі системи та перелік основних функцій, доступних кожному типу користувачів.

Використання даної діаграми є важливим етапом аналізу вимог до програмного забезпечення, оскільки вона дозволяє виявити ключові сценарії використання системи ще до початку програмної реалізації.

Class Diagram

Class Diagram використовується для моделювання структури програмного забезпечення [36] та опису взаємозв’язків між класами системи.

Діаграма класів є однією з основних UML-діаграм при проєктуванні об’єктно-орієнтованих програмних систем.

У межах розроблюваної інформаційно-комунікаційної системи було визначено такі основні класи:

- User;
- Role;
- Client;

- Request;
- Service;
- Log.

Клас User містить інформацію про користувачів системи та включає такі атрибути:

- username;
- email;
- password;
- role;
- created_at.

Клас Client містить персональну інформацію клієнтів:

- ім’я;
- прізвище;
- телефон;
- компанія;
- дата реєстрації.

Клас Request описує заявки користувачів та містить:

- назву заявки;
- опис проблеми;
- статус;
- дату створення;
- дату оновлення.

Між класами реалізовано асоціативні зв’язки.

Наприклад:

- один клієнт може створити багато заявок;
- одна послуга може бути використана в багатьох заявках;
- одна роль може бути пов’язана з багатьма користувачами.

Class Diagram дозволяє:

- сформувати структуру програмного коду;
- визначити атрибути та методи класів;

- реалізувати логіку взаємодії між об’єктами;
- спростити процес розробки серверної частини застосунку.

У процесі реалізації програмного забезпечення UML-діаграма класів безпосередньо використовується для створення моделей Django ORM.

Sequence Diagram

Sequence Diagram призначена для відображення послідовності взаємодії між компонентами системи у часі [36].

Дана діаграма дозволяє моделювати сценарії роботи інформаційної системи та визначати порядок виконання операцій між користувачем, інтерфейсом, серверною логікою та базою даних.

Для розроблюваної системи було сформовано діаграму послідовності для сценарію створення заявки клієнтом.

Основними учасниками процесу є:

- клієнт;
- веб-інтерфейс;
- сервер Django;
- база даних PostgreSQL.

Процес створення заявки складається з таких етапів:

1. Користувач відкриває форму створення заявки.
2. Система відображає веб-форму.
3. Користувач вводить інформацію про проблему.
4. Дані передаються на сервер.
5. Сервер виконує валідацію інформації.
6. Сервер формує SQL-запит через ORM.
7. Дані записуються до бази даних.
8. Система повертає повідомлення про успішне створення заявки.

Sequence Diagram дозволяє:

- аналізувати логіку взаємодії компонентів;
- виявляти потенційні помилки;
- оптимізувати бізнес-процеси;

- моделювати складні сценарії роботи системи.

Особливо важливою дана діаграма є при проєктуванні багатокористувацьких веб-застосунків, де одночасно виконується велика кількість операцій.

Activity Diagram

Activity Diagram використовується для моделювання бізнес-процесів та логіки виконання операцій у системі [36].

Діаграма активності дозволяє описати:

- послідовність дій;
- умови переходів;
- паралельні процеси;
- варіанти завершення сценаріїв.

Для інформаційно-комунікаційної системи було побудовано Activity Diagram процесу обробки заявки.

Основними етапами процесу є:

1. Реєстрація користувача.
2. Авторизація у системі.
3. Створення заявки.
4. Перевірка даних.
5. Передача заявки менеджеру.
6. Аналіз проблеми.
7. Зміна статусу заявки.
8. Завершення консультації.
9. Закриття заявки.

У процесі обробки заявки можуть виникати альтернативні сценарії:

- помилка авторизації;
- відхилення заявки;
- повторне редагування інформації;
- необхідність уточнення даних.

Activity Diagram дозволяє детально описати алгоритм функціонування системи та є важливим інструментом документування бізнес-логіки програмного продукту.

Крім того, дана діаграма значно спрощує подальше тестування системи та перевірку коректності реалізації функціоналу.

Component Diagram

Component Diagram використовується для моделювання архітектури програмного забезпечення та взаємодії між окремими програмними компонентами [36].

Діаграма компонентів відображає:

- серверні модулі;
- клієнтські модулі;
- базу даних;
- зовнішні інтерфейси;
- канали взаємодії між компонентами.

У межах розроблюваної системи виділено такі основні компоненти:

- веб-браузер користувача;
- фронтенд-частина;
- Django-сервер;
- ORM-модуль;
- база даних PostgreSQL;
- модуль безпеки;
- модуль журналювання.

Користувач взаємодіє із системою через браузер. HTTP-запити передаються на серверну частину Django-застосунку, де здійснюється обробка бізнес-логіки.

Django-сервер взаємодіє з базою даних через ORM-механізм, який автоматично формує SQL-запити.

Модуль безпеки виконує:

- автентифікацію;
- авторизацію;

- перевірку сесій;
- контроль доступу.

Модуль журналювання забезпечує аудит подій системи та збереження інформації про дії користувачів.

Component Diagram дозволяє:

- сформувати архітектуру системи;
- визначити межі відповідальності компонентів;
- спростити масштабування;
- забезпечити модульність програмного забезпечення.

Переваги використання UML-діаграм

Використання UML-моделювання під час проєктування інформаційно-комунікаційної системи забезпечує низку важливих переваг [37]:

- стандартизацію процесу розробки;
- спрощення командної взаємодії;
- покращення документування системи;
- зменшення кількості архітектурних помилок;
- спрощення тестування;
- підвищення якості програмного забезпечення.

UML-діаграми дозволяють створити формалізований опис системи ще до початку реалізації програмного коду, що значно знижує ризик виникнення критичних помилок на пізніх етапах розробки.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ CRM-СИСТЕМИ MY_CRM

У цьому розділі наведено опис практичного етапу розробки інформаційно-комунікаційної системи обліку клієнтів та консультацій. Розглянуто особливості програмної реалізації спроєктованої структури бази даних та моделей Django ORM, описано логіку ключових модулів керування заявками, а також налаштування контуру безпеки та розмежування прав користувачів. Наведено результати функціонального тестування та інструкцію користувача для роботи із розробленим веб-застосунком.

4.1 Архітектури програмного забезпечення

Для автоматизації операційного обліку взаємодії з клієнтами розроблено веб-застосунок «my_crm». Програмний комплекс реалізовано на базі високорівневої мови програмування Python [25] та веб-фреймворку Django [26] із використанням архітектурного шаблону MVT (Model-View-Template) [27].

Розподіл функціональних обов'язків між компонентами MVT-патерну у структурі системи визначено таким чином:

- Model (Модель): описує схему даних та правила валідації полів через класи Python. Взаємодіє з реляційними таблицями за допомогою вбудованого механізму об'єктно-реляційного відображення Django ORM.
- View (Представлення): координує бізнес-логіку системи. Приймає вхідні HTTP-запити від користувача, ініціює вибірки з бази даних, перевіряє рольові обмеження доступу та передає сформований контекст даних у шар відображення. У проєкті застосовано класові представлення (Class-Based Views).
- Template (Шаблон): реалізує користувацький інтерфейс за допомогою розмітки HTML5, стилів CSS3 та адаптивної сітки фреймворку Bootstrap [23]. Динамічно рендериться на стороні сервера на основі переданого від View контексту.

Структура директорій розробленого програмного забезпечення має модульну архітектуру (рис. 4.1):

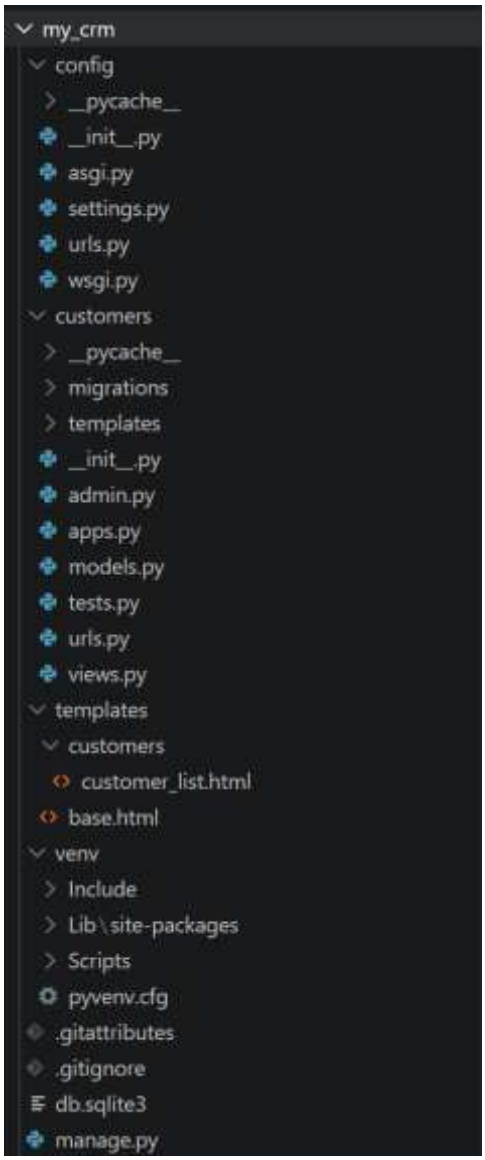

Рис. 4.1 – Загальна структура проєкту

- Пакет config – містить глобальні параметри конфігурації проєкту (settings.py), опис кореневих маршрутів (urls.py) та інтерфейси підключення до веб-сервера (wsgi.py, asgi.py).
- Додаток customers – інкапсулює логіку ведення клієнтської бази та пов'язаних процесів. Включає опис моделей (models.py), контролерів обробки запитів (views.py), конфігурацію форм (forms.py), локальний маршрутизатор (urls.py) та підсистему автоматичних міграцій (migrations/).

- Папка templates – містить загальні HTML-шаблони, зокрема базовий каркас сторінок base.html.

Схема обробки запитів базується на такій послідовності дій: HTTP-запит від браузера надходить на диспетчер URL Dispatcher, який делегує його виконання конкретному класовому представленню у views.py. Останнє, за потреби, здійснює транзакції до файлу БД через ORM, об'єднує отримані об'єкти з відповідним HTML-шаблоном та повертає згенеровану сторінку клієнту (див. рис. 4.2).

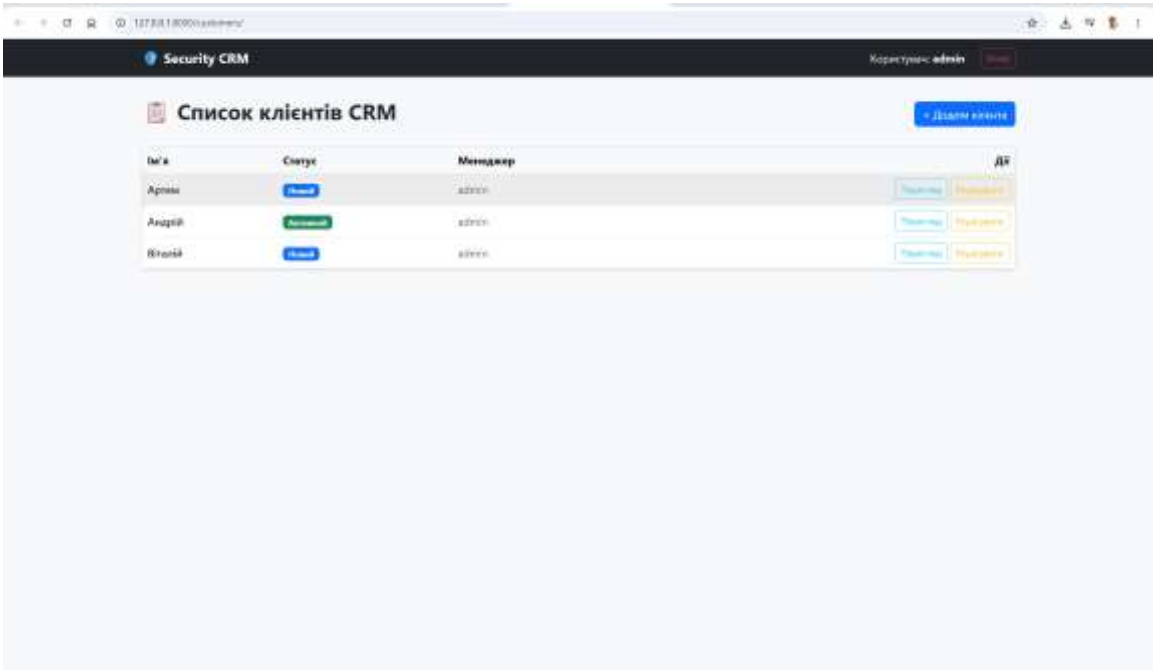

Рисунок 4.2 – Головна сторінка системи

4.2 Реалізація бази даних CRM-системи

Фізичне збереження та обробка інформаційних масивів у межах дипломного проєкту реалізовано за допомогою реляційної СУБД SQLite [28], інтегрованої в екосистему Django. Логічну структуру реляційної моделі даних приведено до третьої нормальної форми (3NF). Сутності, типи даних та обмеження полів на рівні моделей описано в таблицях 4.1 та 4.2.

Таблиця. 4.1 – Специфікація полів відображення інформаційного масиву Customers

Поле	**Опис**	**Тип даних**
id	Integer	Унікальний ідентифікатор
name	Varchar	Ім'я клієнта
email	Varchar	Електронна адреса
phone	Varchar	Номер телефону
status	Varchar	Статус клієнта
manager	ForeignKey	Відповідальний менеджер
created_at	DateTime	Дата створення

Таблиця 4.2 – Специфікація атрибутів моделі даних Customer (Клієнти)

Поле	**Тип даних**	**Опис**
id	Integer	Ідентифікатор угоди
customer	ForeignKey	Клієнт
title	Varchar	Назва угоди
amount	Decimal	Сума угоди
stage	Varchar	Етап продажу
created_at	DateTime	Дата створення

Зв'язок між таблицями Customer та Deal має відношення ForeignKey («один-до-багатьох»), що дозволяє закріплювати за одним клієнтом історію з багатьох послідовних звернень. Синхронізація структури моделей Python з реляційними таблицями SQLite здійснюється через підсистему міграцій командами makemigrations та migrate. Безпека та параметризація запитів контролюються на рівні Django ORM, що унеможливлює виконання несанкціонованого коду через SQL-ін'єкції [26].

4.3 Реалізація функціональних модулів CRM-системи

Бізнес-логіка веб-застосунку розподілена по контролерах у файлі views.py за допомогою спеціалізованих класів:

- CustomerListView — реалізує вибірку та відображення черги клієнтів. Для розмежування прав застосовано перевизначення методу get_queryset(). Якщо користувач має статус is_superuser або is_staff, система повертає повний список записів із БД; операційному менеджеру повертаються виключно ті профілі, де його ідентифікатор вказано у полі manager.
- CustomerDetailView — генерує сторінку картки обраного клієнта, здійснюючи зв'язану вибірку його персональних полів та списку транзакцій з таблиці Deal за допомогою related_name='deals'.
- CustomerCreateView / CustomerUpdateView — обробляють POST-форми створення та модифікації профілів клієнтів з автоматичним валідуванням введених типів даних на відповідність обмеженням моделей.
- DealCreateView — забезпечує реєстрацію нових консалтингових заявок, автоматично підставляючи значення customer_id з URL-маршруту під час збереження об'єкта.

Введення та очищення даних виконуються через інтерфейс Django Forms [26]. Навігаційні переходи у користувацькому інтерфейсі побудовано за принципом мінімізації дій (доступ до ключових функцій у 2–3 кліки) за допомогою системи динамічного формування посилань URL Routing. Керування користувачами та системними довідниками типів послуг винесено у вбудовану адміністративну панель Django Admin (admin.py) [30].

4.4 Реалізація безпеки та адміністрування CRM-системи

Захист контуру інформаційної системи та оброблюваних персональних даних реалізовано наступним комплексом інженерних рішень:

- Обмеження доступу: Усі функціональні класи-контролери успадковують міксин LoginRequiredMixin. Це блокує прямий доступ

неавторизованих користувачів до внутрішніх URL-адрес системи, автоматично перенаправляючи запити на форму автентифікації.

- Розмежування привілеїв: Рольова модель (RBAC) розділяє права на рівні бізнес-логіки. Менеджер обмежений роботою із закріпленою за ним базою клієнтів та створенням угод, тоді як операції видалення та глобального налаштування дозволені лише адміністратору.
- Криптографічний захист: Паролі користувачів проходять обов'язкове одностороннє хешування за алгоритмом PBKDF2 з додаванням випадкової криптографічної солі, унеможливлюючи відновлення первинних символів у разі компрометації файлу БД.
- Захист від веб-атак: Міжсайтова підробка запитів нейтралізується обов'язковою перевіркою CSRF-токенів ({% csrf_token %}) для кожної форми [26]. Спроби міжсайтового скриптингу (XSS) блокуються за рахунок автоматичного екранування текстових змінних у шаблонізаторі.
- Цілісність даних: Регулярне резервне копіювання інформаційного масиву реалізується копіюванням цілісного файлу бази даних db.sqlite3 після зупинки сервера застосунку. Спроби несанкціонованого підбору паролів чи системні збої фіксуються вбудованими інструментами журналювання подій.

4.5 Тестування CRM-системи та інструкція користувача

Після завершення розробки CRM-системи my_crm було проведено комплексне тестування програмного забезпечення. Основною метою тестування було підтвердження працездатності функціональних модулів, перевірка коректності роботи механізмів авторизації, взаємодії з базою даних та оцінка стабільності системи в різних сценаріях використання.

Тестування є невід'ємною складовою процесу розробки програмного забезпечення [29], оскільки дозволяє виявити помилки ще до впровадження системи в експлуатацію.

Для перевірки працездатності було сформовано набір тестових сценаріїв, наведений у таблиці 4.5.

Таблиця 4.5 – Таблиця тестових сценаріїв

№	**Тестовий сценарій**	**Очікуваний результат**
1	Авторизація користувача	Успішний вхід до системи
2	Невірний пароль	Повідомлення про помилку
3	Створення клієнта	Клієнт додається до бази
4	Редагування клієнта	Дані оновлюються
5	Перегляд клієнта	Відображення картки клієнта
6	Створення угоди	Угода зберігається
7	Перегляд угод	Відображення списку угод
8	Вхід до адміністративної панелі	Доступ лише для адміністратора
9	Робота з базою даних	Коректне збереження даних
10	Завершення сесії	Успішний вихід із системи

Результати тестування показали коректну роботу всіх основних функціональних можливостей системи.

Позитивне та негативне тестування: Під час позитивного тестування перевірялася робота системи за умови введення коректних даних. Вхід із валідними обліковими даними пройшов успішно, ініціювавши активну сесію з редіректом на список клієнтів (див. рис. 4.3). Модулі створення клієнта (рис. 4.4),

редагування даних (рис. 4.5) та реєстрації нових угод успішно виконали транзакції у базі даних.

Негативне тестування підтвердило стійкість системи до помилок: введення неправильного пароля заблокувало вхід із виведенням попередження. Механізми валідації полів Django Forms зупинили спроби надсилання порожніх обов'язкових форм. Спроби неавторизованого доступу до сторінки /customers/ були заблоковані міксином LoginRequiredMixin із перенаправленням на сторінку входу.

Загальні підсумки верифікації системи зведені у таблиці 4.6.

Таблиця 4.6 – Результати функціональної верифікації компонентів системи

Перевірка	**Результат**
Авторизація	Успішно
Робота сесій	Успішно
Створення клієнтів	Успішно
Редагування клієнтів	Успішно
Створення угод	Успішно
Захист доступу	Успішно
Робота бази даних	Успішно
Адміністративна панель	Успішно

Результат тестування підтвердив правильність роботи механізму авторизації (див. рис. 4.3).

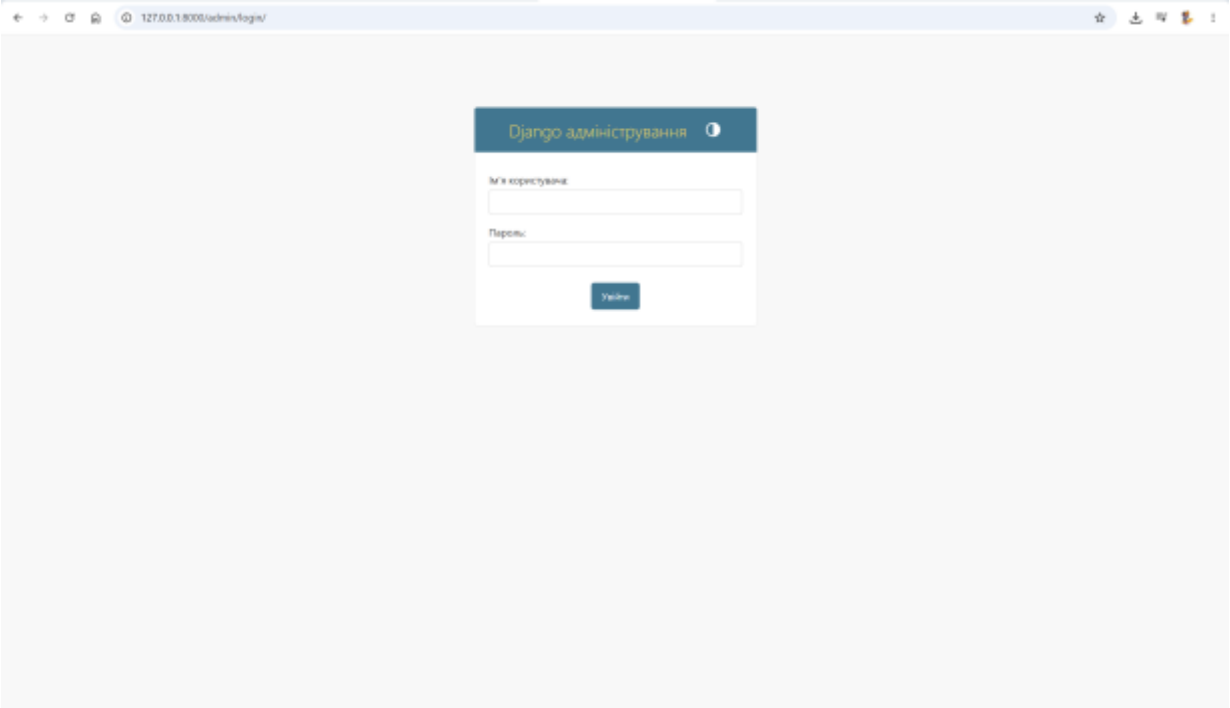

Рисунок 4.3 – Сторінка авторизації

Створення нового клієнта

Було введено коректні дані нового клієнта.

Очікуваний результат (рис. 4.4):

- створення запису;
- збереження інформації у базі даних;
- відображення нового клієнта у списку.

Система успішно виконала всі необхідні операції.

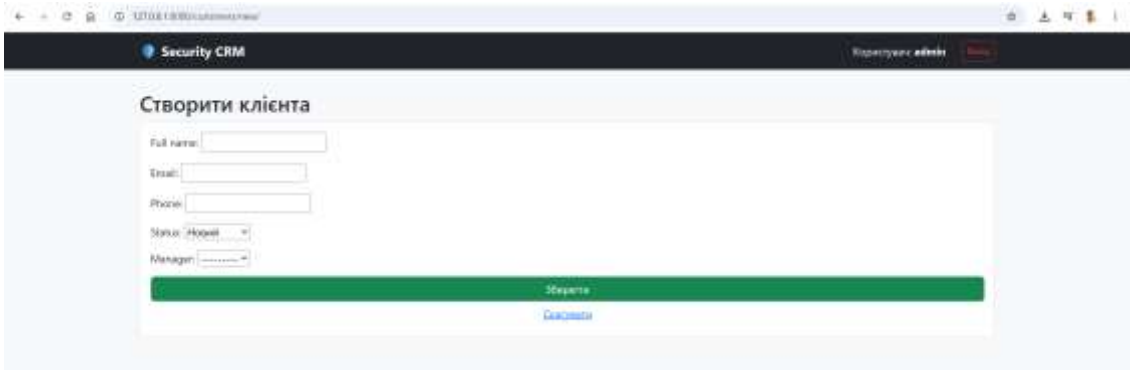

Рисунок 4.4 – створення нового клієнта

Редагування інформації про клієнта

Було змінено номер телефону та статус клієнта.

Очікуваний результат (рис. 4.5):

- оновлення запису;
- збереження змін у базі даних.

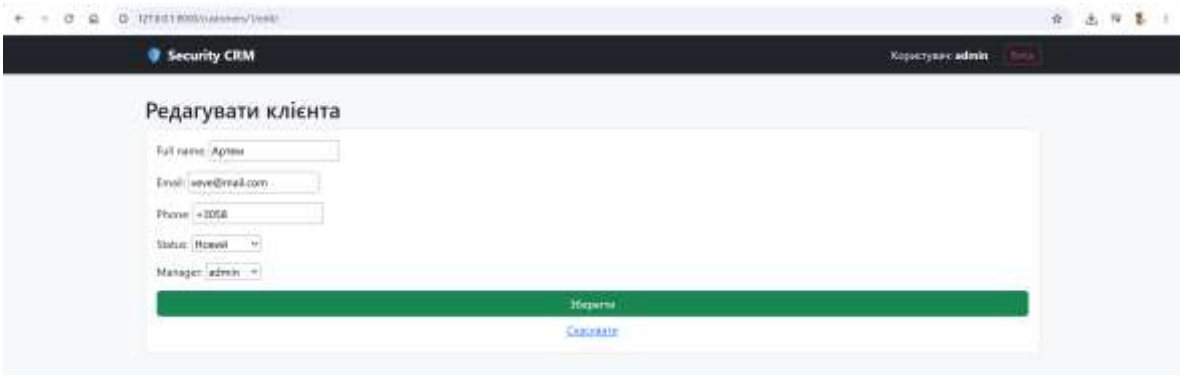

Рисунок 4.5 – Редагування інформації про клієнта

Тестування підтвердило правильність роботи механізму редагування.

Створення угоди

Було створено нову угоду для існуючого клієнта.

Очікуваний результат:

- збереження угоди;
- прив'язка до клієнта;
- відображення на сторінці клієнта.

Система коректно виконала всі операції.

Інструкція користувача

Для запуску веб-застосунку необхідно відкрити командний рядок у каталозі проєкту, активувати віртуальне середовище та виконати команду запуску вбудованого сервера розробки Django (див. рис. 4.6):

```
(venv) PS D:\Коди та проекти\my_crm> python manage.py runserver
```

Рисунок 4.6 – запуск сервера

Після цього система стає доступною у браузері за адресою http://127.0.0.1:8000/ (рис. 4.7). Первинне створення профілю адміністратора виконується утилітою python manage.py createsuperuser з урахуванням введення логіна, e-mail та стійкого пароля.

Рисунок 4.7 – адреса сайту

Створення адміністратора

Для створення адміністратора необхідно виконати команду (рис. 4.8):

```
python manage.py createsuperuser
```

Рисунок 4.8 – створення адміністратора

Після введення логіна, електронної пошти та пароля буде створено адміністративний обліковий запис.

Вхід до системи

Для авторизації необхідно відкрити сторінку входу, ввести логін, ввести пароль та натиснути кнопку входу.

Після успішної авторизації користувач отримує доступ до функціональних можливостей CRM-системи.

Авторизація користувачів здійснюється через стандартну форму введення облікових даних (рис. 4.3). Основне меню системи забезпечує роботу з такими модулями:

1. Ведення бази клієнтів: перегляд таблиці контрагентів з їхніми контактами, статусами (Proposal, Negotiation, Closed Won, Closed Lost) та відповідальними менеджерами. Додавання та коригування даних виконується через екранні форми «Додати клієнта» та «Редагувати».

2. Керування угодами: у картці конкретного клієнта за допомогою кнопки «Нова угода» реалізовано створення консалтингової заявки із зазначенням суми та поточного етапу надання послуги.

3. Адміністрування: повний доступ до налаштувань, логів безпеки, груп прав та облікових записів персоналу здійснюється адміністратором за адресою http://127.0.0.1:8000/admin (див. рис. 4.9).

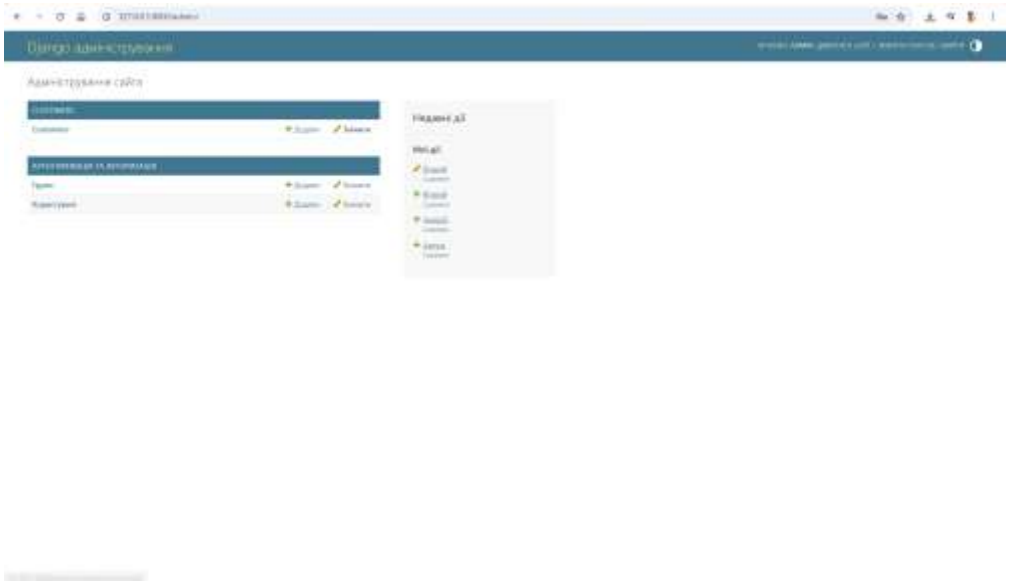

Рисунок 4.9 – Використання адміністративної панелі

Це значно спрощує процес адміністрування CRM-системи.

ВИСНОВКИ

У дипломній роботі виконано повний комплекс дослідницьких, проєктних та програмних робіт, спрямованих на вирішення актуальної інженерної задачі — автоматизації операційного обліку взаємодії з клієнтами та управління процесом надання консультаційних послуг з кібербезпеки в умовах сервісної IT-компанії.

За результатами виконання роботи зроблено такі основні висновки:

1. Аналіз предметної області та обґрунтування актуальності. Досліджено специфіку діяльності сервісних компаній та доведено, що використання неавтоматизованих підходів призводить до високих ризиків втрати даних та зниження швидкості обслуговування. Виявлено, що сфера кібербезпеки вимагає підвищеного рівня конфіденційності та суворого розмежування доступу до інформації про вразливості клієнтів, що обґрунтовує необхідність створення спеціалізованого програмного забезпечення.

2. Проєктні рішення. На основі методологій інженерії ПЗ розроблено функціональні та нефункціональні вимоги до системи. За допомогою уніфікованої мови моделювання UML побудовано діаграму прецедентів (Use Case) та діаграму послідовностей (Sequence Diagram), що дозволило чітко формалізувати архітектурні межі та життєвий цикл обробки HTTP-запитів. Спроєктовано реляційну структуру бази даних, яку приведено до третьої нормальної форми (3NF), що унеможливлює виникнення аномалій та надмірності даних.

3. Обґрунтування технологій. Аргументовано вибір високорівневої мови програмування Python та веб-фреймворку Django, архітектура якого базується на шаблоні MVT (Model-View-Template) та містить потужний інструмент ORM і вбудований контур безпеки. Для надійного збереження зв'язаних даних обрано реляційну СУБД, а для розробки адаптивного клієнтського інтерфейсу — стек HTML5/CSS3/Bootstrap/JavaScript.

4. Програмна реалізація та безпека. Програмно реалізовано веб-застосунок, структуровано класи моделей даних, налаштовано автоматичну підсистему міграцій та генерацію інтерфейсів. Особливу увагу приділено безпеці:

впроваджено рольову модель доступу (RBAC) з трьома ролями (Клієнт, Менеджер, Адміністратор), реалізовано кастомні декоратори для валідації прав користувачів, а також криптографічне хешування паролів за алгоритмом PBKDF2 з сіллю та захист від критичних веб-уразливостей (SQLi, CSRF, XSS).

5. Тестування та практична цінність. Проведено функціональне тестування, тестування безпеки та UI-компонентів за допомогою розроблених тестових сценаріїв (Test Cases). Результати перевірки підтвердили повну працездатність бізнес-логіки системи, стійкість до несанкціонованого підвищення привілеїв та високу швидкість обробки даних. Створений програмний продукт повністю готовий до експлуатації та має високу практичну цінність для малих і середніх сервісних ІТ-підприємств.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Software Engineering. URL: https://www.goodreads.com/en/book/show/103569.Software_Engineering
2. Software Engineering: A Practitioner's Approach. URL: https://www.mlsu.ac.in/econtents/16_EBOOK-7th_ed_software_engineering_a_practitioners_approach_by_roger_s._pressman_.pdf
3. Django Documentation. URL: https://docs.djangoproject.com/en/6.0/
4. Database System Concepts. URL: https://www.db-book.com/
5. Role-Based Access Control Models. Ravi S. Sandhu', Edward J. Coynek, Hal L. Feinsteink and Charles E. Youman. URL: https://csrc.nist.gov/csrc/media/projects/role-based-access-control/documents/sandhu96.pdf
6. OWASP Top 10. URL: https://owasp.org/www-project-top-ten/
7. Google Workspace Admin Help. Access control and sharing settings. URL: https://support.google.com/a/
8. Bitrix24. CRM Platform Features. URL: https://www.bitrix24.com/
9. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. URL: https://www.javier8a.com/itc/bd1/articulo.pdf
10. Pressman R., Maxim B. Software Engineering: A Practitioner's Approach. McGraw-Hill. URL: https://books.google.com.ua/books?id=bL7QZHtWvaUC&lpg=PA1&ots=O9B7aRuP5k&dq=Pressman%20R.%2C%20Maxim%20B.%20Software%20Engineering%3A%20A%20Practitioner's%20Approach.%20McGraw-Hill.&lr&hl=ru&pg=PA1#v=onepage&q&f=false
11. Django Documentation – MTV Design Pattern. URL: https://docs.djangoproject.com/en/stable/faq/general

12. Sandhu R., Coyne E., Feinstein H., Youman C. Role-Based Access Control Models. IEEE Computer, 1996. URL: https://csrc.nist.gov/csrc/media/projects/role-based-access-control/documents/sandhu96.pdf
13. OWASP Password Storage Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
14. OWASP Transport Layer Security Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Security_Cheat_Sheet.html
15. OWASP SQL Injection Prevention Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html
16. OWASP Cross Site Request Forgery Prevention Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html
17. OWASP Cross Site Scripting Prevention Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
18. Laravel Documentation. URL: https://laravel.com/docs/13.x
19. Node.js Documentation. URL: https://nodejs.org/docs/latest/api/
20. Django Documentation. URL: https://docs.djangoproject.com/en/6.0/
21. Database System Concepts. McGraw-Hill Education. URL: https://www.db-book.com/db6/index.html
22. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/
23. Bootstrap Documentation. URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/
24. MDN Web Docs JavaScript Guide. URL: https://developer.mozilla.org/

25. Sommerville I. Software Engineering. – 10th ed. – Pearson, 2016. URL: https://www.studyhalo.com/media/resources/resources/INF3705/Textbook/Software_Engineering_-_Ian_Sommerville.pdf
26. Sandhu R., Coyne E., Feinstein H., Youman C. Role-Based Access Control Models // IEEE Computer. – 1996. URL: https://profsandhu.com/journals/computer/i94rbac(org).pdf
27. Pressman R., Maxim B. Software Engineering: A Practitioner's Approach. – McGraw-Hill, 2019. URL: https://www.mlsu.ac.in/econtents/16_EBOOK-7th_ed_software_engineering_a_practitioners_approach_by_roger_s._pressman_.pdf
28. OWASP Password Storage Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
29. Django Documentation. Authentication System. URL: https://docs.djangoproject.com/en/6.0/topics/auth/default/
30. Django Documentation. The Django Admin Site. URL: https://docs.djangoproject.com/en/6.0/ref/contrib/admin/
31. Django Documentation. Models and Databases. URL: https://docs.djangoproject.com/en/6.0/topics/db/
32. MDN Web Docs. JavaScript Guide. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
33. Silberschatz A., Korth H., Sudarshan S. Database System Concepts. URL: https://www.mpgcamb.com/wp-content/uploads/2024/12/Abraham-Silberschatz-Henry-F.-Korth-S.-Sudarshan-Database-System-Concepts-McGraw-Hill-Education-2019.pdf
34. Elmasri R., Navathe S. Fundamentals of Database Systems. URL: https://www.uoitc.edu.iq/images/documents/informatics-institute/Competitive_exam/Database_Systems.pdf
35. Date C. J. An Introduction to Database Systems. URL: https://lc.fie.umich.mx/~rodrigo/BD/An%20Introduction%20to%20Database%20Systems%208e%20By%20C%20J%20Date.pdf

36. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. URL: https://martinfowler.com/books/uml.html
37. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. URL: https://www.uml.org/

ДОДАТКИ

Додаток А. Програмний код

1. my_crm\customers\admin.py

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from django.contrib.auth.models import User
from .models import Customer

# Налаштування для моделі Клієнтів
@admin.register(Customer)
class CustomerAdmin(admin.ModelAdmin):
    list_display = ('full_name', 'email', 'status',
'manager')
    list_filter = ('status', 'manager')

    # Додаємо цей рядок для появи пошуку
    search_fields = ('full_name', 'email', 'phone')

# Ваші існуючі налаштування для Користувачів
class MyUserAdmin(UserAdmin):
    def get_queryset(self, request):
        qs = super().get_queryset(request)
        if not request.user.is_superuser:
            return qs.filter(is_superuser=False)
        return qs

admin.site.unregister(User)
admin.site.register(User, MyUserAdmin)
```

2. my_crm\customers\apps.py

```
from django.apps import AppConfig


class CustomersConfig(AppConfig):
    name = 'customers'
```

3. my_crm\customers\models.py

```
    from django.db import models
    from django.contrib.auth.models import User

    class Customer(models.Model):
        STATUS_CHOICES = [
            ('new', 'Новий'),
            ('active', 'Активний'),
            ('lost', 'Втрачений'),
        ]
        full_name = models.CharField(max_length=255)
        email = models.EmailField(unique=True)
        phone       =       models.CharField(max_length=20,
blank=True)
        status      =      models.CharField(max_length=10,
choices=STATUS_CHOICES, default='new')
        manager          =          models.ForeignKey(User,
on_delete=models.SET_NULL, null=True)
        created_at                                          =
models.DateTimeField(auto_now_add=True)

        def __str__(self):
            return self.full_name

    class Deal(models.Model):
        STAGES = [
            ('proposal', 'Пропозиція'),
            ('negotiation', 'Переговори'),
            ('closed_won', 'Закрито (Успіх)'),
            ('closed_lost', 'Закрито (Відмова)'),
        ]

        customer       =       models.ForeignKey(Customer,
on_delete=models.CASCADE, related_name='deals')
        name = models.CharField(max_length=200)
        amount     =     models.DecimalField(max_digits=10,
decimal_places=2)
        stage       =       models.CharField(max_length=20,
choices=STAGES, default='proposal')
        created_at                                          =
models.DateTimeField(auto_now_add=True)

        def __str__(self):
            return f"{self.name} - {self.amount} грн"
```

4. my_crm\customers\views.py

```
    from django.contrib.auth.mixins import
LoginRequiredMixin
    from django.views.generic import ListView, DetailView,
CreateView, UpdateView
    from django.urls import reverse_lazy
    from .models import Customer, Deal   # Переконайтеся,
що Deal тут додано

    class CustomerListView(LoginRequiredMixin, ListView):
        model = Customer
        template_name = 'customers/customer_list.html'
        context_object_name = 'customers'

        def get_queryset(self):
            if      self.request.user.is_superuser      or
self.request.user.is_staff:
                return Customer.objects.all()
            return
Customer.objects.filter(manager=self.request.user)

    class            CustomerDetailView(LoginRequiredMixin,
DetailView):
        model = Customer
        template_name = 'customers/customer_detail.html'

    class            CustomerCreateView(LoginRequiredMixin,
CreateView):
        model = Customer
        fields = ['full_name', 'email', 'phone', 'status',
'manager']
        template_name = 'customers/customer_form.html'
        success_url = reverse_lazy('customer_list')

    class            CustomerUpdateView(LoginRequiredMixin,
UpdateView):
        model = Customer
        fields = ['full_name', 'email', 'phone', 'status',
'manager']
        template_name = 'customers/customer_form.html'
        success_url = reverse_lazy('customer_list')
```

```
    # ПЕРЕВІРТЕ ВІДСТУПИ ТУТ (4 пробіли перед кожним
рядком всередині класу)
    class DealCreateView(LoginRequiredMixin, CreateView):
        model = Deal
        template_name = 'customers/deal_form.html'
        fields = ['name', 'amount', 'stage']

        def form_valid(self, form):
            form.instance.customer_id = self.kwargs['pk']
            return super().form_valid(form)

        def get_success_url(self):
            return          reverse_lazy('customer_detail',
kwargs={'pk': self.kwargs['pk']})
```

5. my_crm\config\settings.py

```
    """
    Django settings for config project.
    """

    from pathlib import Path

    # Build paths inside the project like this: BASE_DIR /
'subdir'.
    BASE_DIR = Path(__file__).resolve().parent.parent


    # Quick-start development settings - unsuitable for
production
    #                                                 See
https://docs.djangoproject.com/en/6.0/howto/deployment/chec
klist/

    # SECURITY WARNING: keep the secret key used in
production secret!
    SECRET_KEY              =              'django-insecure-
wnl@og+g0!kn8!p9h!)qbtty2k_n2j+k93tv@p+@*c1s+l2!wr'

    # SECURITY WARNING: don't run with debug turned on in
production!
    DEBUG = True

    ALLOWED_HOSTS = []
```

```
    # Application definition

    INSTALLED_APPS = [
        'django.contrib.admin',
        'django.contrib.auth',
        'django.contrib.contenttypes',
        'django.contrib.sessions',
        'django.contrib.messages',
        'django.contrib.staticfiles',

        # Власний додаток для CRM
        'customers',
    ]

    MIDDLEWARE = [
        'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',
        'django.middleware.common.CommonMiddleware',
        'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',
    ]

    ROOT_URLCONF = 'config.urls'

    TEMPLATES = [
        {
            'BACKEND':
'django.template.backends.django.DjangoTemplates',
            'DIRS': [BASE_DIR / 'templates'], # Шлях до
глобальних шаблонів
            'APP_DIRS': True,
            'OPTIONS': {
                'context_processors': [

'django.template.context_processors.request',
```

```
'django.contrib.auth.context_processors.auth',

'django.contrib.messages.context_processors.messages',
                ],
            },
        },
    ]

    WSGI_APPLICATION = 'config.wsgi.application'


    # Database
    #
https://docs.djangoproject.com/en/6.0/ref/settings/#databas
es

    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': BASE_DIR / 'db.sqlite3',
        }
    }


    # Password validation
    #
https://docs.djangoproject.com/en/6.0/ref/settings/#auth-
password-validators

    AUTH_PASSWORD_VALIDATORS = [
        {
            'NAME':
'django.contrib.auth.password_validation.UserAttributeSimil
arityValidator',
        },
        {
            'NAME':
'django.contrib.auth.password_validation.MinimumLengthValid
ator',
        },
        {
```

```
            'NAME':
'django.contrib.auth.password_validation.CommonPasswordVali
dator',
        },
        {
            'NAME':
'django.contrib.auth.password_validation.NumericPasswordVal
idator',
        },
    ]


    # Internationalization
    #  Налаштовуємо українську мову та київський час для
CRM

    LANGUAGE_CODE = 'uk'

    TIME_ZONE = 'Europe/Kyiv'

    USE_I18N = True

    USE_TZ = True


    # Static files (CSS, JavaScript, Images)
    #   https://docs.djangoproject.com/en/6.0/howto/static-
files/

    STATIC_URL = 'static/'
    STATICFILES_DIRS = [BASE_DIR / 'static']

    # Default primary key field type
    DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

    # Налаштування авторизації
    LOGIN_URL = '/admin/login/'
    LOGIN_REDIRECT_URL = '/customers/'
    LOGOUT_REDIRECT_URL = '/admin/login/'
```

6. my_crm\config\urls.py

```
    from django.contrib import admin
```

```
    from django.urls import path, include
    from django.views.generic import RedirectView # Додайте цей імпорт

    urlpatterns = [
        path('admin/', admin.site.urls),
        path('customers/', include('customers.urls')),

        # Це правило перенаправить головну сторінку на список клієнтів
        path('', RedirectView.as_view(url='/customers/', permanent=True)),
    ]
```

7. my_crm\config\wsgi.py

```
    """
    WSGI config for config project.

    It exposes the WSGI callable as a module-level variable named ``application``.

    For more information on this file, see
    https://docs.djangoproject.com/en/6.0/howto/deployment/wsgi/
    """

    import os

    from django.core.wsgi import get_wsgi_application

    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'config.settings')

    application = get_wsgi_application()
```

8. my_crm\config\asgi.py

```
    """
    ASGI config for config project.

    It exposes the ASGI callable as a module-level variable named ``application``.
```

```
    For more information on this file, see
    https://docs.djangoproject.com/en/6.0/howto/deployment
/asgi/
    """

    import os

    from django.core.asgi import get_asgi_application

    os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'config.settings')

    application = get_asgi_application()
```

9. my_crm\templates\base.html

```
    <!DOCTYPE html>
    <html lang="uk">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1.0">
        <title>♡□ CRM Система - Дипломний проєкт</title>
        <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css
/bootstrap.min.css" rel="stylesheet">
        <style>
            body { background-color: #f8f9fa; }
            .navbar { margin-bottom: 2rem; box-shadow: 0
2px 4px rgba(0,0,0,0.1); }
        </style>
    </head>
    <body>
        <nav class="navbar navbar-expand-lg navbar-dark
bg-dark">
            <div class="container">
                <a class="navbar-brand fw-bold"
href="/">♡□ Security CRM</a>

                <div class="navbar-nav ms-auto align-
items-center">
                    {% if user.is_authenticated %}
```

```
                        <span class="nav-link text-light
me-3">
                            Користувач: <strong>{{
user.username }}</strong>
                        </span>

                        <form action="/admin/logout/"
method="post" class="d-inline">
                            {% csrf_token %}
                            <button type="submit"
class="btn btn-outline-danger btn-sm">Вихід</button>
                        </form>
                    {% else %}
                        <a class="btn btn-outline-primary
btn-sm" href="/admin/login/">Увійти</a>
                    {% endif %}
                </div>
            </div>
        </nav>

        <div class="container">
            {% block content %}{% endblock %}
        </div>

        <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/b
ootstrap.bundle.min.js"></script>
    </body>
    </html>
```

10. my_crm\templates\customers\customer_list.html

```
    {% extends 'base.html' %}

    {% block content %}
    <div class="d-flex justify-content-between align-
items-center mb-4">
        <h2 class="fw-bold">📋 Список клієнтів CRM</h2>
        <a href="{% url 'customer_create' %}" class="btn
btn-primary">+ Додати клієнта</a>
    </div>

    <div class="card shadow-sm">
        <div class="card-body p-0">
```

```
            <table class="table table-hover mb-0">
                <thead class="table-light">
                    <tr>
                        <th>Ім'я</th>
                        <th>Статус</th>
                        <th>Менеджер</th>
                        <th class="text-end">Дії</th>
                    </tr>
                </thead>
                <tbody>
                    {% for customer in customers %}
                    <tr>
                        <td     class="align-middle     fw-
semibold">{{ customer.full_name }}</td>
                        <td class="align-middle">
                            {%   if   customer.status   ==
'active' %}
                                <span   class="badge   bg-
success">Активний</span>
                            {%   elif   customer.status   ==
'new' %}
                                <span   class="badge   bg-
primary">Новий</span>
                            {% else %}
                                <span   class="badge   bg-
secondary">Втрачений</span>
                            {% endif %}
                        </td>
                        <td    class="align-middle    text-
muted">{{ customer.manager.username }}</td>
                        <td class="text-end">
                            <a         href="{%         url
'customer_detail'  customer.pk  %}"  class="btn  btn-sm  btn-
outline-info">Перегляд</a>
                            <a         href="{%         url
'customer_edit'  customer.pk  %}"  class="btn  btn-sm  btn-
outline-warning">Редагувати</a>
                        </td>
                    </tr>
                    {% empty %}
                    <tr>
                        <td colspan="4" class="text-center
py-4">Клієнтів не знайдено</td>
                    </tr>
```

```
                    {% endfor %}
                </tbody>
            </table>
        </div>
    </div>
    {% endblock %}
```

11. my_crm\customers\templates\customers\customer_detail.html

```
    {% extends 'base.html' %}
    {% block content %}
    <div class="mt-4">
        <div class="d-flex justify-content-between">
            <h2>{{ customer.full_name }}</h2>
            <a href="{% url 'deal_create' customer.pk %}"
class="btn btn-outline-primary">+ Нова угода</a>
        </div>
        <p>Емейл: {{ customer.email }} | Статус: {{
customer.get_status_display }}</p>

        <h3 class="mt-4">Активні угоди</h3>
        <table class="table">
            <thead>
                <tr>
                    <th>Назва</th>
                    <th>Сума</th>
                    <th>Етап</th>
                </tr>
            </thead>
            <tbody>
                {% for deal in customer.deals.all %}
                <tr>
                    <td>{{ deal.name }}</td>
                    <td>{{ deal.amount }} грн</td>
                    <td><span class="badge bg-warning
text-dark">{{ deal.get_stage_display }}</span></td>
                </tr>
                {% empty %}
                <tr><td colspan="3">Угод ще
немає.</td></tr>
                {% endfor %}
            </tbody>
        </table>
```

```
        <a href="{% url 'customer_list' %}" class="btn
btn-secondary">Назад до списку</a>
    </div>
    {% endblock %}
```

12. my_crm\customers\templates\customers\customer_form.html

```
    {% extends 'base.html' %}

    {% block content %}
    <h2>{% if object %}Редагувати{% else %}Створити{%
endif %} клієнта</h2>
    <form method="post" class="card card-body">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="btn btn-
success">Зберегти</button>
        <a href="{% url 'customer_list' %}" class="btn
btn-link">Скасувати</a>
    </form>
    {% endblock %}
```

13. my_crm\customers\templates\customers\customer_list.html

```
    {% extends 'base.html' %}


    {% block content %}
    <div class="d-flex justify-content-between mb-3">
        <h2>Список клієнтів</h2>
        <a href="{% url 'customer_create' %}" class="btn
btn-primary">+ Додати клієнта</a>
    </div>


    <table class="table table-hover">
        <thead class="table-light">
            <tr>
                <th>Ім'я</th>
                <th>Статус</th>
                <th>Дії</th>
            </tr>
        </thead>
```

```
        <tbody>
            {% for customer in customers %}
            <tr>
                <td><a href="{% url 'customer_detail'
customer.pk %}">{{ customer.full_name }}</a></td>
                <td><span class="badge bg-info text-
dark">{{ customer.get_status_display }}</span></td>
                <td>
                    <a href="{% url 'customer_edit'
customer.pk %}" class="btn btn-sm btn-outline-
secondary">Редагувати</a>
                </td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
    {% endblock %}
```

14. my_crm\customers\templates\customers\deal_form.html

```
    {% extends 'base.html' %}
    {% block content %}
    <h2>Додати нову угоду</h2>
    <form method="post" class="card card-body shadow-sm">
        {% csrf_token %}
        {{ form.as_p }}
        <button     type="submit"     class="btn     btn-
primary">Створити угоду</button>
    </form>
    {% endblock %}
```

15. my_crm\customers\urls.py

```
    from django.urls import path
    from . import views

    urlpatterns = [
        path('',          views.CustomerListView.as_view(),
name='customer_list'),
        path('<int:pk>/',
views.CustomerDetailView.as_view(),
name='customer_detail'),
        path('new/',    views.CustomerCreateView.as_view(),
name='customer_create'),
```

```
        path('<int:pk>/edit/',
views.CustomerUpdateView.as_view(), name='customer_edit'),
        # Цей рядок виправляє помилку NoReverseMatch
        path('<int:pk>/deal/new/',
views.DealCreateView.as_view(), name='deal_create'),
    ]
```

16. my_crm\manage.py

```
    #!/usr/bin/env python
    """Django's command-line utility for administrative
tasks."""
    import os
    import sys


    def main():
        """Run administrative tasks."""
        os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'config.settings')
        try:
            from       django.core.management       import
execute_from_command_line
        except ImportError as exc:
            raise ImportError(
                "Couldn't import Django. Are you sure it's
installed and "
                "available on your PYTHONPATH environment
variable? Did you "
                "forget    to    activate    a    virtual
environment?"
            ) from exc
        execute_from_command_line(sys.argv)


    if __name__ == '__main__':
        main()
```

Додаток Б. Посилання на репозиторій

Посилання:

https://github.com/ArtemB704/my_crm